SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

D I P L O M S K I R A D

Darko Dokladal

Zagreb, 2012.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

D I P L O M S K I R A D

Mentor:
Prof. dr. sc. Mario Essert

Student:
Darko Dokladal

Zagreb, 2012.

FAKULTET STROJARSTVA I BRODOGRADNJE

Povjerenstvo za diplomske i završne ispite

IZJAVA

Pod punom moralnom odgovornošću izjavljujem da sam rad radio samostalno koristeći se znanjem stečenim tijekom studija, te navedenom literaturom.

Zahvaljujem se prof. dr. sc. Mariu Essertu na ukazanom povjerenju prihvaćanjem mentorstva za ovaj rad i korisnim savjetima.

Veliko hvala mojim roditeljima što su mi omogućili školovanje, te svima onima koji su me podržavali tijekom studija, a posebno mojoj djevojci Kristini.

Darko Dokladal

Zagreb, 2012.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

Središnje povjerenstvo za završne i diplomske ispite

Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **Darko DOKLADAL** Mat. br.: 0035161754

Naslov rada na hrvatskom jeziku: **Program za prepoznavanje znakova pomoću invarijantnih momenata**

Naslov rada na engleskom jeziku: **Character Recognition Program Based on Invariant Moments**

Opis zadatka:

U znanstvenom području obrade slika i vizijskih sustava provode se opsežna istraživanja izvlačenja značajki oblika i prepoznavanje objekta. Princip prepoznavanja temelji se na poklapanju opisanih oblika. Razvijen je velik broj tehnika koje matematički opisuju oblike, a među njima se razlikuju skalarne značajke (dimenzije, površina, broj kutova itd.), *Fourierove* značajke i invarijantni momenti. Te tehnike numerički opisuju oblik neovisno o translaciji, mjerilu i rotaciji i mogu jednostavno biti primjenjene na prepoznavanje objekata, znakova itd. Invarijantni momenti postali su bitan alat kad je riječ o prepoznavanju i identifikaciji.

Za realizaciju diplomskog zadatka potrebno je:

1. Zamisliti i realizirati programsko sučelje za obradu i procesiranje slike;
2. Opisati i primjeniti centralne geometrijske momente, kompleksne momente, te afine momente u opisivanju i prepoznavanju oblika;
3. Testirati i prikazati uspješnost pojedinih značajki (deskriptora) u prepoznavanju oblika;
4. Opisati i primjeniti algoritme strojnog učenja, te uključiti ih s invarijantnim momentima u prepoznavanju oblika;
5. Dokumentaciju provesti u *HTML/PDF* oblicima uporabom alata *Sphinx*.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
4. listopada 2012.	6. prosinca 2012.	12. – 14. prosinca 2012.

Zadatak zadao: Predsjednik Povjerenstva:

M. Essert

Prof.dr.sc. Mario Essert

Prof. dr. sc. Franjo Cajner

Sadržaj

Popis slika

Popis tablica

Popis oznaka i mjernih jedinica

m_{pq}	općí moment
$f(x, y)$	karakteristična funkcija slike
c_{pq}	centralni kompleksni moment
μ_{pq}	centralni geometrijski moment
ϕ_n	Hu-ov moment

Sažetak

Cilj ovog diplomskog rada je provjeriti uspješnost invarijantnih momenata kao opisivača oblika u svrhu prepoznavanja znakova. Kako bi to bilo moguće potrebno je učiniti čitav niz predradnji. Sučelje programa mora imati mogućnost interakcije s korisnikom kako bi se izvršavale određene manipulacije slike. Dakle, prije nego što možemo vršiti bilo kakvo prepoznavanje potrebno je sliku segmentirati odnosno izdvojiti objekte od interesa sa slike. Objekti su pohranjeni u bazu zajedno s ostalim informacijama o objektu. Program je realiziran u programskom jeziku Python. Python je visoki programski jezik opće namjene, a glavni naglasak je na čitljivosti kôda i brze realizacije zamišljenih ideja. Za grafičko sučelje programa korišten je Pythonov modul *Wx* kojim je jednostavno napraviti robusno i visokofunkcionalno sučelje. Operacije prikaza i manipulacije slikama ostvarene su OpenCV-om. OpenCV je dobro poznat i već dugo prisutan alat za računalni vid. Pomoću modula za strojno učenje *sklearn* realizirano je prepoznavanje znakova i oblika.

Ključne riječi: invarijantni momenti, prepoznavanje znakova, prepoznavanje oblika

UVOD

U našem svakodnevnom životu procesuiramo i analiziramo veliku količinu informacija različitih vrsta, značaja i kvalitete te na temelju analize tih informacija donosimo odluke. Informacije dobivamo kroz osjetila, a više od 95% njih su optičke prirode. Slike su moćan medij za prijenos informacija i sredstvo komuniciranja. Zbog toga, ne samo da su primarni izvor informacija već i način komuniciranja između ljudi i strojeva, npr. na računalu imamo monitor koji služi kao sredstvo prijenosa slike odnosno informacije što se događa u računalu. Digitalne fotografije sadrže veliku količinu informacija. Zato se razvijaju posebne metode za analizu slika. Analiza i interpretacija slika uzetih neidealnim sustavom (fotoaparat, kamera) glavni je problem u mnogim područjima kao što su robotski vid, astronomija, medicina itd. Kako su sustavi za uzimanje fotografija, kao i uvjeti u kojima se fotografija uzima, nesavršeni tako dobijemo degradiranu verziju promatrane scene. Na degradaciju slike mogu utjecati razni čimbenici poput geometrije scene, aberacija leće, fokus, brzina objekata u sceni, sistematične i slučajne pogreške senzora itd. Pod pojmom „analiza“ obično podrazumijevamo kompleksno procesiranje slike koje se sastoji od dvije faze. Prva faza, slika se predprocesira, segmentira te se obilježe potencijalni objekti od interesa. Druga faza, objekti se prepoznaju, što znači da su matematički opisani i klasificirani kao objekti određene klase između drugih preddefiniranih klasa.

Jedan od osnovnih problema pri stvaranju sustava za prepoznavanje uzoraka tiče se odabira prikladnih numeričkih atributa značajki koje je potrebno izvući iz slike u svrhu klasifikacije. Matematički koncept momenata je prisutan već desecima godina i koristi se u granama znanosti poput mehanike i statistike pa i prepoznavanja oblika. Opisivanje slike momentima umjesto drugim uobičajenim značajkama slike znači koristiti globalne osobine slike umjesto lokalnih.

Hu je bio prvi koji je, 1961. godine, objavio značajan rad o upotrebi invarijantnih momenata u svrhu analize slike i prepoznavanje objekata. Hu-ov rad temeljio se na radu teorije algebri invarijantnih formi matematičara Boolea, Cayleya i Sylvestera iz devetnaestog stoljeća.

1.1 Značaj prepoznavanje uzorka

Prepoznavanje uzorka je znanstvena disciplina čiji je cilj klasifikacija objekta u određen broj kategorija ili klasa. Ovisno o vrsti aplikacije, taj objekt mogu biti slike ili valni signala ili bilo koji tip mjerljivih osobina koje treba klasificirati. Takve objekte nazivat ćemo generičkim nazivom uzorak. Povijest prepoznavanja uzorka je dugačka, ali prije 1960tih godina sastojala se uglavnom od teoretskih istraživanja u području statistike. Vrijeme računala donijelo je i potrebu za praktičkim aplikacijama prepoznavanja uzorka, što je napravilo podlogu za novim teoretskim razmatranjima i razvoju. Kako je društvo napredovalo iz industrijskog u postindustrijsko doba, automatizacija u industrijskoj proizvodnji i potreba upravljanja i dohvata informacija dobilo je značajnu važnost. Takav trend je gurnuo prepoznavanje uzorka do visokog stupnja na kojem se danas nalazi u aplikacijama i istraživanju. Prepoznavanje uzorka je integrirani dio sistema s umjetnom inteligencijom ugrađen za donošenje odluka. Vizijski sustavi su područje gdje prepoznavanje uzorka je od velike važnosti. Vizijski sustav prikupljenu sliku (preko kamere) analizira kako bi dobio opis onoga što se nalazi na njoj. Tipičan primjer upotrebe vizijskog sustava je u tvornicama, bilo kao automatska vizualna inspekcija ili automatizacija montažne linije. Na primjer, u inspekciji, proizvedeni objekt na pomičnom nosaču proizvoda prolazi pokraj stanice za ispitivanje, gdje je smještena kamera, te treba utvrditi postoji li defekt. Stoga, slika mora biti analizirana u realnom vremenu, a sustav prepoznavanja uzorka mora klasificirati objekt kao „greška“ ili „nema greške“. Nakon toga potrebno je izvršiti radnju odbacivanja dijela ako on uistinu nije dobar. Na liniji za montažu, različiti dijelovi moraju biti locirani i prepoznati, odnosno klasificirani u jednu od klasa, prije nego je njihova upotreba moguća, kako bi robotska ruka znala pristupiti predmetu na točnu lokaciju i upotrijebiti ga uz pomoć pravog alata. Prepoznavanje znakova (slova i brojeva) je još jedno bitno područje prepoznavanje uzorka, gdje dolazi do izražaja automatizacija i upravljanje informacijama. Sustavi za optičko prepoznavanje znakova (eng. Optical Character Recognition – OCR) su već dugi niz godina komercijalno dostupni te u velikoj mjeri poznati svima nama. Nakon što OCR sustav prikupi sliku ona je transformirana u „brojeve“ te spremljena kao polje. U nastavku, primjenjuje se niz tehnika procesiranja slike koji vode do segmentacije linija i znakova. Zatim sustav za prepoznavanje uzoraka izvršava prepoznavanje znakova, a to je klasificiranje svakog znaka u točno određenu klasu „znak, broj, inpuktacija“. Spremanje dokumenata koji su prošli proces prevođenja ima dvostruku prednost u odnosu na spremanje skeniranih dokumenata. Prvo, daljnja obrada, ako je potrebna, je lako moguća kroz programe za obradu teksta, a drugo, memorijski je puno efikasnije spremiti dokument u obliku ASCII znakova nego slikovnog dokumenta. Pored sustava za prepoznavanje računalom tiskanih znakova, značajan trud je uložen u prepoznavanje rukom pisanih znakova. Tipična komercijalna primjena takve aplikacije bila bi bankama kao provjera rukopisa pri kupnji, gdje je kupnju potrebno autorizirati potpisom. Još jedna primjena mogla bi biti u uredima pošte kao automatsko sortiranje pošte prema poštanskom broju. U današnjim pa-

metnim telefonima nalaze se upravo takvi sustavi koji prepoznaju pisane znakove koje korisnik uz pripadajuću olovku ispisuje po ekranu osjetljivom na dodir, što može biti efikasnije i brže pri pisanju bilješki. Računalom pomognuta dijagnostika je druga važna aplikacija prepoznavanja uzorka čiji je cilj pomoći doktorima u donošenju dijagnostičkih odluka. Konačna odluka je, naravno, doktorova. Računalom pomognuta dijagnostika je primjenjiva u raznim medicinskim podacima, kao što su rendgenske slike, slike računalne tomografije, slike ultrazvuka, kardiodijagrami, elektroencelofalogrami (EEG). Potreba za uvođenjem računalom pomognutu dijagnostiku proizlazi iz činjenice da je medicinske podatke često teško interpretirati, i interpretacija može ovisiti individualno o vještini doktora. Prepoznavanje govora je također područje koje je postalo izrazito popularno u današnjim uređajima. Ulaže se puno truda u istraživanje i razvoj, a rezultati su danas implementirani u pametnim telefonima. Govor je ljudima najprirodniji način komunikacije i izmjene informacija. Stoga je dugogodišnji cilj za inženjere i znanstvenike bio izgradnja inteligentnih strojeva koji prepoznaju izgovorenu informaciju. Potencijalne primjene su brojne. Naravno kako bi se postigao konačni cilj prepoznavanju uzorka u svim navedenim aplikacijama, prepoznavanje uzorka je usko povezano s drugim znanstvenim disciplinama kao što su lingvistika, računalna grafika i vizija.

1.2 Segmentacija

Pri prepoznavanju oblika na slici uvijek je bitno točno odrediti što zapravo želimo prepoznati, odnosno izolirati objekt na slici koji je od interesa. Iako ovisi koji alat se koristi za opisivanje oblika segmentacija je postupak particioniranja slike u regije. Postupak segmentacije se izvodi pomoću vizijskog algoritma, te se regije tretiraju kao kandidati za dijelove objekata. Osnovna karakteristika regije je da pikseli koji čine regiju zadovoljavaju nekakav uvjet ili funkciju, za razliku od svih ostalih piksela na slici. Kvaliteta rješenja ovisi o funkciji po kojoj se izvodi segmentacija. Skupina povezanih piksela sa sličnim svojstvima čini jednu regiju, dok se unutar prizora može nalaziti više objekata, a ovisno o algoritmu koji se koristi i složenosti geometrije objekata u prizoru, pojedini objekt može biti prikazan s više regija.

Algoritme za segmentaciju dijelimo u dvije grupe:

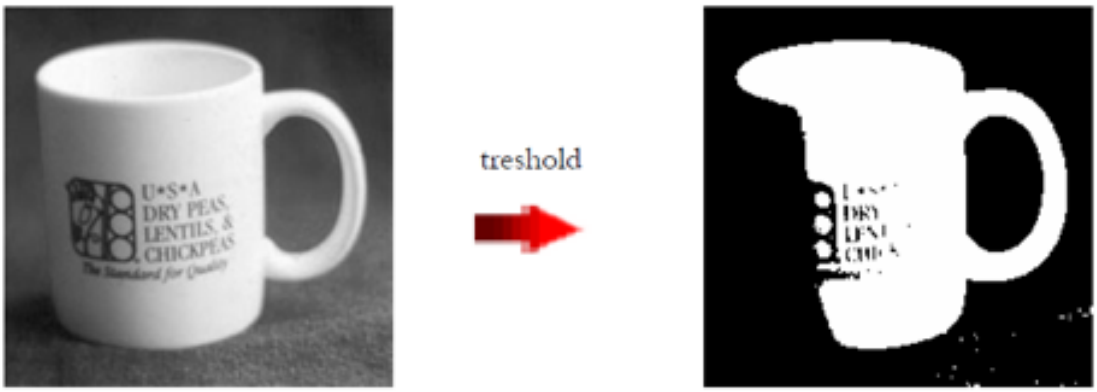
- algoritmi za segmentaciju bazirani na regijama,
- algoritmi za segmentaciju bazirani na bridovima.

Kod segmentacije bazirane na regijama, pikseli se dodjeljuju u skupinu na temelju kriterija koji ih razlikuje od ostalih piksela na slici tj. označava se skupina piksela koja odgovara objektu. Definiranjem bridova i upotrebom jednostavnog algoritma za popunjavanje kontura mogu se dobiti regije koje bi trebale odgovarati objektu. Alternativni pristup je particioniranje konturama tako da konture predstavljaju granice regija. Način identifikacije je preko razlike inten-

ziteta osvijetljenosti susjednih piksela. Algoritmi za praćenje ruba produciraju prikaz regija konturama, pa su oba rješenja kompatibilna, ali će rijetko rezultati biti isti zbog nesavršenosti algoritma.

1.3 Threshold algoritmi

Najjednostavniji način segmentacije je upotreba treshold algoritma koji monokromatske ili multispektralne slike pretvara u binarne, te na taj način odvaja objekt od pozadine. Učinkovitost algoritma ovisi o razlici kontrasta između objekta i pozadine. Kontrast mora biti dovoljno velik. Segmentacija upotrebom treshold algoritma prikazana je na slici. Iz rješenja su vidljivi problemi koji se javljaju kod particioniranja slike. Svijetli objekt se nalazi na tamnoj podlozi, te bi objekt trebao biti odvojen od okoline. Uzrok je sjena i/ili nepravilno definiran prag pri kojoj pikseli postaju crni odnosno bijeli.



Slika 1.1: Primjer threshold segmentacije.

1.3.1 Segmentacija pomoću treshold algoritma

Ne postoji univerzalna tehnika segmentacije koja će raditi na svim slikama niti postoji savršena tehnika segmentacije. Gray-level thresholding je najjednostavniji segmentacijski proces. Mnogi objekti ili regije slike imaju karakterističnu konstantnu refleksiju ili apsorpciju svjetla na njihovim površinama – to nam omogućuje da odredimo neki konstantni iznos tj. prag koji će razdvajati objekte od pozadine. Određivanje praga je računski nezahtjevna i jednostavna metoda. To je najstarija metoda koja još ima široku primjenu za jednostavnije zadatke te se vrlo jednostavno izvodi u realnom vremenu uz pomoć adekvatne opreme. Vrlo je bitno odrediti dobar prag jer je samo u rijetkim slučajevima više vrijednosti praga dobro za cijelu sliku

(globalni prag) pošto već i kod dosta jednostavnih slika dolazi do varijacija – te varijacije mogu biti uzrokovane nejednakim osvjetljenjem i nizom drugih faktora.

1.4 Algoritmi detekcije rubova

Segmentacija bazirana na rubovima se zasniva na rubovima pronađenim uz pomoć raznih detektora. Ti rubovi označavaju lokacije diskontinuiteta između nijansi, boja, teksture ili nečeg drugoga. Najčešće se problemi kod ove segmentacije javljaju zbog šumova ili drugih vrsta loših informacija o slici – npr., da se registrira rub gdje ne postoji ili da se ne registrira gdje on postoji. Granice objekata se mogu izdvojiti metodama: praćenje granice, interpolacije krivulja ili Hough-ovom transformacijom.

1.5 Algoritmi temeljeni na regijama slike

Algoritmi temeljeni na regijama je metoda identificira dijelove slike koje imaju slične značajke. Za razliku od prethodnih metoda u kojima su se tražili rubovi između regija ove metode direktno određuju regije. Ova metoda je pogodnija za slike koje imaju dosta šuma i gdje je teško odrediti granice. Dvije karakteristične metode su: izrastanje područja (eng. region growing), metoda dijeljenja i stapanja (eng. split and merge). Evaluacija segmentacije je korisna kada trebamo izabrati jedan od više algoritama ili odabrati parametre za izabrani algoritam.

1.6 Načini prepoznavanja

Prepoznavanje objekta i uzoraka na slikama koje su na neki način iskrivljene je tema mnogih istraživanja. U principu postoje tri načina da bi se to postiglo – metoda sirovom snagom, normalizacijom slike, te invarijatnim značajkama. Pristupu prepoznavanja sirovom snagom parametriziramo sve moguće degradacije slike. To znači da set podataka za učenje mora sadržati rotiranu, skaliranu, zamučenu i ostale verzije deformacije objekta kojeg želimo klasificirati. Jasno, takav pristup vodi do velikih zahtjeva u procesorianju podataka i praktički je neupotrebljiv. Normalizacijom slike, objekt se transformira u određeni standardni položaj prije nego se proslijedi klasifikatoru. Takav pristup je učinkovit u dijelu klasifikacije ali normalizacija objekta obično zahtjeva rješavanje kompleksnih inverznih problema. Korištenje invarijantnih značajki čini se kao pristup koji ima najviše perspektive i opsežno se koristi. Osnovna ideja koja stoji iza toga je da se objekt opiše nizom mjerljivih kvantiteta koje nazivamo invarijantama koje su nepromjenjive na određene deformacije te pružaju dovoljnu raznolikost da bi se razlikovali objekti koji pripadaju različitim klasama. S matematičkog gledišta, invarijanta I je

funkcional, definiran funkcijom slike, koji ne mijenja svoju vrijednost pod utjecajem degradacijskog operatora D. Tako mora biti zadovoljen uvjet I(f) = I(D(f)) za svaku funkciju f. To svojstvo se naziva invarijanta. U praksi, kako bi priskočili utjecaju nesavršenosti segmentacije, varijaciji unutar određene klase i šumu, obično formuliramo zahtjev kao slabije odnosno blaže ograničenje: I(f) bi trebao biti sličan I(D(f)). Slijedeća poželjna osobina invarijanti je diskriminabilnost, koja je jednako važna kao i invarijanca. Ta dva zahtjeva međusobno su kontradiktorna. Što je šira granica invarijance, manja je mogućnost raspoznavanja odnosno manja je diskriminacija. Pravilnim odabirom optimalnih vrijednosti invarijance i diskriminativnosti je bitan zadatak u prepoznavanju objekata. Obično jedna invarijanta nije dovoljna da bi dobili dovoljno diskriminacije stoga se koristi nekoliko invarijanti istovremeno. U tom slučaju imamo invarijantni vektor, te je svaki objekt prezentiran kao točka u n-dimenzionalnim prostoru.

1.7 Vrste invarijanti

Slijedi kratak pregled vrsti invarijanti te njihova kategorizacija. Invarijante kojima opisujemo 2D objekte mogu se podijeliti na različite načine s različitih gledišta. Najjednostavnija podjela invarijanti je prema tipu invarijance. Tako poznajemo translacijske, rotacijske, skalirane, afine, projekcijske invarijance. Invarijante također možemo podijeliti prema tipu matematičkog alata koji se koristi:

- Jednostavni opisivači oblika (eng. simple shape descirptors)
- Značajke koeficijentima (eng. transform coefficient features)
- Invarijante točkama (eng. point set invariants)
- Diferencijalne invarijante (eng. differential invariants)
- Invarijantni momenti (eng. moment invariants)

Slijedeći pregled daje podjelu prema potrebnom dijelu objekta da bi se izračunala invarijanta. Tako poznajemo globalne, lokalne i polu-lokalne invarijante. Globalne invarijante se računaju iz cijele slike (uključujući pozadinu ako segmentacija nije izvršena). U usporedbi s lokalnim invarijantama, globalne invarijante su puno robusnije na šum, netočno prepoznavanje granica objekta i sl. S druge strane, nedostatak globalnih invarijanti je što lokalna promjena slike utječe na vrijednost svih invarijanti, te ta promjena nije lokaliziran u nekoliko komponenti. Iz tog razloga globalne invarijante ne mogu se koristiti kada je promatrani objekt djelomično prikriven drugim objektom ili je dio objekta van vidnog polja. Invarijantni momenti spadaju u ovu kategoriju. Lokalne invarijante se izračunavaju iz određenih točaka na ograničenom prostoru. Diferencijalne invarijante su tipičan predstavnik te kategorije. Prvo se detektiraju granice objekta, a zatim se izračunaju invarijante za svaku točku koja omeđuje objekt. Kao rezultat

dobije se da invarijante ovise samo o obliku granice u svojoj blizini. Ako ostatak objekta prođe kroz neku promjenu, lokalne invarijante nisu promijenjene. Ove osobine čine ih odličnim alatom za prepoznavanje objekata koji su dijelom zaklonjeni, no zbog velike osjetljivosti na greške diskretizacije, netočnost segmentacije i šum, teško ih je u praksi primijeniti.

Polulokalne invarijante su pokušaj objedinjavanja dobrih osobina, a pri tome izbjeći negativne, od gore navedenih grupa. Ova vrsta invarijanti pokušava objekt podijeliti na dijelove (obično je podjela na mjestima pregiba ili sl.) i opisati ih nekom vrstom globalnih invarijanti. Cijeli objekt je tada opisan nizom vektora invarijanti te se prepoznavanje u uvjetima djelomično prikrivenog objekta obavlja pretraživanjem niza vektora te maksimalnim poklapanjem podniza.

U ovom radu fokus će biti na opisu i prepoznavanju objekata pomoću invarijantnih momenata. Invarijantni momenti osmišljeni su još prije pojave prvih računala, u 19. stoljeću u istraživanju teorije algebre invarijanti. Invarijantni momenti prvi put su upotrjebljeni u svrhu prepoznavanja uzoraka i procesiranja slike 1962. godine, kada je Hu predstavio rezultate teorije algebre invarijanti i izveo svojih sedam poznatih invarijanti za rotaciju 2D objekta. Od tada, stotine znanstvenih radova je bilo posvećeno raznim unaprjeđenjima, proširenjima i generalizaciji invarijantnih momenata kao i njihovim raznim aplikacijama u mnogim područjima. Invarijantni momenti su postali jedni od najpopularnijih alata za opisivanje oblika (eng. shape descriptors). Iako imaju problema zbog određenih ograničenja (najveći je njihova globalnost, koja sprječava prepoznavanje zaklonjenih objekata) i dalje su čest prvi izbor, a znaju služiti i kao referentna metoda za procjenu performansi drugih opisivača oblika. Unatoč enormnom trudu i velikom broju članaka objavljenih na tu temu, brojni problemi ostali su neriješeni.

PYTHON

Python je visoki programski jezik opće namjene, a glavni naglasak je na čitljivosti kôda. Cilj programskog jezika Python je jednostavnu sintaksu, a njegovi standardni moduli su laki za korištenje. Interpreter, interaktivan, objektno orijentiran, iznimke, visoki dinamički tipovi podataka i klase (razredi) sve su to odlike Python jezika. Python je lako prenosiv na druge platforme kao što su Unix, MacOS i Windows. U ovom radu Python je korišten kao sredstvo realizacije zadatka.

2.1 Standardni Python moduli

2.1.1 Sučelje s operativnim sustavom

Modul `os` sadrži funkcije za interakciju s operativnim sustavom:

```
>>> import os
>>> os.getcwd()      # vraćena vrijednost je trenutni radni direktorij
'C:\\Python27'
>>> os.chdir('/server/accesslogs')   # promjena radnog direktorija
>>> os.system('mkdir xyz')   # izvršavanje naredbe 'mkdir xyz'
0
```

Preporuča se korištenje `import os` umjesto `from os import *` stila jer `os.open()` će zasjeniti ugrađenu funkciju `open()` koja radi na drugom principu.

Ugrađene funkcije `dir()` i `help()` su korisne ako zatreba brza pomoć pri radu s velikim modulima poput `os`:

```
>>> import os
>>> dir(os)
<vraća listu svih funkcija modula>
>>> help(os)
<vraća tekst generiran iz modulovog stringa za dokumentaciju>
```

Za rad s datotekama modul `shutil` pruža sučelje koje olakšavaju manipulaciju njima:

```
>>> import shutil
>>> shutil.copyfile('podaci.db', 'arhiva.db')
>>> shutil.move('/dir/dir1', 'dir2')
```

2.1.2 Rad s datotekama

Modul `glob` sadrži funkciju koja vraća listu datoteka koje odgovaraju zadanim uvjetima pretrage:

```
>>> import glob
>>> glob.glob('*.py')
['primarni.py', 'cijeli.py', 'konfiguracija.py']
```

2.1.3 Argumenti naredbene linije

Skripte pisane u Pythonu često trebaju procesirati argumente zadane iz naredbene linije pri njenom pokretanju. Argumenti su spremljeni u atributu *argv* modula `sys`. Pokretanjem linije `python demo.py jedan dva tri`, sljedeći kôd ispisati će argumente zadane pri pokretanju, primjer:

```
>>> import sys
>>> print sys.argv
['demo.py', 'jedan', 'dva', 'tri']
```

Modul `getopt` koristi Unixov princip funkcije `getopt()`. Više mogućnosti i fleksibilnosti pri baratanju takvom vrstom argumenata da je modul `argparse`.

2.1.4 Upozorenja i greške

Modul `sys` također sadrži atribute za *stdin*, *stdout*, i *stderr*. Posljednji se koristi pri izdavanju poruka upozorenja i greški:

```
>>> sys.stderr.write('Greška, pokušaj ponovno\n')
Greška, pokušaj ponovno
```

Direktan način kako skripta može prekinuti izvođenje je upotrebom `sys.exit()` naredbe.

2.1.5 Pretraživanje znakovnih nizova

Modul `re` sadrži alate za manipulaciju regularnim izrazima koji služe pri procesiranju znakovnih nizova. Regularni izrazi predstavljaju optimizirano rješenje za takve manipulacije:

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'glavni fokus Arch Linux
                  distribucije je jednostavnost')
['fokus']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'glavni fokus Arch Linux
                  distribucije je jednostavnost')
'glavni fokus Arch Linux distribucije je jednostavnost'
```

Pri jednostavnim manipulacijama lakše je koristiti metode znakovnih nizova zbog veće čitljivosti kôda i uklanjanja greški:

```
>>> 'primjer za dvojcu'.replace('dvojcu', 'dvoje')
'primjer za dvoje'
```

2.1.6 Matematika u Pythonu

Modul `math` daje pristup funkcijama iz jezika C za matematičke operacije s pomičnim zarezom:

```
>>> import math
>>> math.cos(math.pi / 4.0)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

`random` modul ima alate za generiranje slučajnih brojeva i slično:

```
>>> import random
>>> random.choice(['jabuka', 'sir', 'orah'])
'jabuka' # slučajno izbačen znakovni niz
>>> random.sample(xrange(100), 10)   # slučajno generiran niz brojeva
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()    # slučajano generiran broj
0.17970987693706186
>>> random.randrange(6)    # slučajan cijeli broj u rasponu do (6)
4
```

2.1.7 Pristup Internetu

Postoje nekoliko modula za pristup internetu i procesiranje internet protokola. Dva najjednostavnija su `urlib2` za dobivanje podataka kroz web adrese i `smtplib` za slanje e-maila:

```
>>> import urllib2
>>> for line in urllib2.urlopen('http://tycho.usno.navy.mil
                                        /cgi-bin/timer.pl'):
...     if 'EST' in line or 'EDT' in line:  # traži Eastern Time
...         print line

<BR>Nov. 30, 10:31:14 AM EST Eastern Time

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('primjer1@primjer.org', 'primjer2@primjer.org',
... """To: primjer2@primjer.org
... From: primjer1@primjer.orgs
... LP.
... """)
>>> server.quit()
```

(Da bi pokrenuli drugi primjer, na računalu mora postojati mail server.)

2.1.8 Datum i vrijeme

Modul `datetime` pruža razrede za datum i vrijeme u jednostavnoj i naprednoj verziji. Postoje i moduli koji podržavaju vrijeme s obzirom na vremensku zonu.

```
>>> # datumi se jednostavno
>>> from datetime import date
>>> danasnji_datum = date.today()
>>> danasnji_datum
datetime.date(2012, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-12. 02 Dec 2003 is a Sunday on the 02 day of December.'
>>> # podržana je i arigmetika datumima
>>> rodjendan = date(1964, 7, 31)
>>> godine = danasnji_datum - rodjendan
>>> godine.days
17656
```

2.1.9 Kompresija podataka

Arhiviranje podataka i kompresija su direktno podržani u modulima `zlib`, `gzip`, `bz2`, `zipfile` i `taffile`.:

```
>>> import zlib
>>> s = 'Glavni fokus Arch Linux distribucije je jednostavnost'
>>> len(s)
41
>>> t = zlib.compress(s)
>>> len(t)
37
>>> zlib.decompress(t)
'Glavni fokus Arch Linux distribucije je jednostavnost'
>>> zlib.crc32(s)
1747727218
```

2.1.10 Mjerenje performansi

Ponekad je interesantno ili potrebno relativne performanse različitih pristupa istom problemu. Python sadrži i takav alat.

Primjer `timeit` modula:

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

Moduli `profil` i `pstats` mogu identificirati vrijeme kritičnih dijelova i velikih blokova kôda.

2.1.11 Kontrola kvalitete kôda

Jedan od načina pri razvijanju programa je da se pišu testovi za svaku funkciju razvijenu, te ispitivanjem tih funkcija kroz proces razvoja. Takav pristup razvoju programa rezultira visokom kvalitetom programa zbog minimiziranja mogućnosti greški.

`doctest` modul svojim alatima skenira modul i testira dijelove kôda koji su pod znacima komentara. Dijelovi kôda predstavljaju kako želimo pozvati funkciju koju testiramo s pripadajućim argumentima, te kakav rezultat očekujemo od funkcije. Testiranje se odvija kao da poziv

funkcije vršimo iz komandne linije. Na takav način napisan program poboljšava dokumentaciju i čitljivost zbog uvida u primjere poziva i rezultate.

```
def prosjek(vrijednosti):
    """Prosjek svih brojeva u nizu.

    >>> print prosjek([20, 30, 70])
    40.0
    """
    return sum(vrijednosti, 0.0) / len(vrijednosti)

import doctest
doctest.testmod()    # automatski radi test kôda
```

2.1.12 Raznolikost Pythona

Python pruža apsolutno sve, a to najbolje dolazi do izražaja pogledom na njegov velik broj paketa. Pa tako:

- `xmlrpclib` i `SimpleXMLRPCServer` moduli implementiraju poziv funkcije s udaljenog računala i čine to prilično jednostavnim. Iako moduli u imenu imaju 'XML', XML kao takav nije nužno poznavati za njihovo korištenje.
- modul `email` je paket za manipulaciju e-mail porukama. `smptlib` i `poplib` koji šalju i primaju poruke, email paket ima cjelokupni alat za baratanje strukturom poruke (uključujući i privitke) i enkodiranje zaglavlja protokola.
- `xml.dom` i `xml.sax` paketi daju robusnu podršku pri parsiranju XML-a. Pa tako `cvs` modul podržava čitanje i pisanje u bazu. Svi ti alati zajedno doprinose lakšoj manipulaciji podacima i njihovoj tranziciji između drugih aplikacija.

Ovo su neki od primjera kako Python stvarno ima podršku za sve, te ne treba imati temeljito znanje o korištenju određene tehnologije već samo njene osnove da bi sve radilo kako treba.

RAZREDI U PYTHONU

Programiranje je dobilo novu dimenziju uvođenjem razreda i objekta. Pisanje velikih programa postalo je sistematičnije i lakše. Kako Python sve što se u njemu događa gleda kao objekte ovdje će biti opisani razredi u programskom jeziku Python. U usporedbi s ostalim programskim jezicima, Python ima sličan mehanizam funkcioniranja razreda (ili klasa) s minimalnim izmjenama u sintaksi i semantici. Implementacije razreda programskih jezica C++ i Modula-3 su mješavina razreda u Pythonu. Pythonove klase pružaju sve standardne značajke objektnog programiranja: nasljeđivanjem je moguće izvesti novi razred uz pomoć više baznih razreda, izvedeni razred može imati metodu istog imena kao razred ili razredi iz kojeg je izveden. Objekti nekog razreda mogu sadržavati proizvoljnu količinu različitih podataka. Jednako kao i za različite module Pythona, razredi doprinose dinamičnoj prirodi programskog jezika Python: objekti se stvaraju u vrijeme izvođenja programa, a u daljnjem korištenju se mogu modificirati.

C++ terminologijom, članovi razreda (uključujući članove podataka) su podrazumijevano *javni*, a svi članovi funkcija su *virtualni*. Kao u Modula-3 jeziku, ne postoje nedostaci za referenciranje članova objekta u metodi istog razreda: metoda se deklarira s eksplicitnim prvim argumentom koji predstavlja objekt, a prosljeđuje se implicitno pri pozivu funkcije. Razredi su također objekti, kao u programskom jeziku Smalltalk. Jedno od osnovnih svojstva C++ jezika je enkapsulacija pojam koji označava objedinjavanje podataka i operacija. Jednako je i u Pythonu, većina ugrađenih operatora (aritmetički operatori i sl.) mogu se predefinirati za potrebe objekta razreda.

U nedostatku opće prihvaćene terminologije kad je riječ o razredima, povremeno će biti korišteni pojmovi C++-a i Smalltalka.

3.1 Imena i objekti

Objekti posjeduju svojevrsnu individualnost, više imena može biti povezano s istim objektom. Taj efekt je poznat pod imenom aliasing. Konkretno, aliasing opisuje situaciju kada jednoj memorijskoj lokaciji možemo pristupiti kroz više simboličnih imena u programu. Promjenom podatka kroz jedno ime izmijenit će podatak povezan sa svim imenima, što može biti neočeki-

vano od strane programera. Aliasing se može zanemariti pri korištenjem nepromjenjivih tipova (brojevi, stringovi, tuple). No, alisaing ima drugačiji efekt u semantici Pythonovog kôda pri korištenju promjenjivih objekata kao što su liste, rječnici i ostali tipovi. Razlog je brzina programa jer se aliasing efekt ponaša kao pokazivač. Npr., prosljeđivanje objekta kao argument nekoj funkciji ne zahtjeva puno resursa jer Pythonova implementacija će proslijedi samo pokazivač na taj objekt, mjesto gdje je funkcija pozvana će vidjeti promjenu na objektu. Ovakav sustav eliminira potrebu za mehanizmom gdje postoje dva različita prosljeđivanja argumenta kao što je u Pascalu.

3.2 Domene i imenici u Pythonu

Pri upoznavanju s razredima prvo ćemo reći nešto o Pythonovim pravilima domena imena. Da bi se radile naprednije stvari u Pythonu potrebno je znati ponešto o domenama (eng. scope) i imenicima (eng. namespaces) kako bi se u potpunosti moglo shvatiti što se uistinu događa.

Definicije:

Imenik je mapa između imena i objekata. Većina imenika je već učitano u Pythonov riječnik, ali to nije zamjetno osim u vidu performansi. Primjer imenika su ključne riječi (funkcije kao što su `abs()`); globalna imena i moduli, te lokalna imena pri pozivu funkcija. Bitno je spomenuti da ne postoji nikakva veza između imena u različitim imenicima, npr. dva modula mogu koristiti istu funkciju `max` bez ikakve kolizije — programer mora koristiti prefiks kao ime modula pri korištenju funkcije.

Atribut je termin koji se koristi za ime koje je iza točke — npr. u izrazu `z.realno`, `realno` je atribut objekta `z`. Referenciranje imena u modulu je referenciranje atributa, u izrazu `modul.funkcija`, `modul` je modul objekta a `funkcija` je atribut njegov.

Atributi mogu biti promjenjivi i nepromjenjivi. Ako su promjenjivi, moguće je provesti operaciju dodjeljivanja. Većina atributa je promjenjiva pa je moguće pisati ovako: `modname.rezultat = 42`. Promjenjivi atributi mogu biti izbrisani pomoću ključne riječi *del*. Npr., `del modname.rezultat` će maknuti atribut `rezultat` iz objekta `modname`.

Imenici se stvaraju u različitim trenucima i imaju različit vijek trajanja. Imenici koji sadrže ključne riječi programskog jezika stvaraju se pri pokretanju Python interpretera i ne brišu se nikada. Globalni imenik modula stvara se pri čitanju definicija modula, a također traje do izlaska iz interpretera. Naredbe koje se izvode bilo kroz interpreter interaktivno ili zapisano u skriptu smatraju se dijelom modula imena `__main__`.

Lokalni imenik funkcije je stvoren pri pozivu funkcije i briše se kada funkcija vrati vrijednost ili izbaci iznimku koja nije kontrolirana unutar funkcije. Rekurzivni pozivi funkcije stvara svaki

svoj imenik.

Domena je tekstualna regija Python programa gdje je imenik direktno dostupan. Iako se domene određuju statički, koriste se dinamično. U bilo kojem trenutku izvođenja, postoje barem tri domene čiji su imenici direktno dostupni.

Ako je ime deklarirano globalno tada sve reference i dodjele idu direktno kroz srednju domenu koja sadrži modulova globalna imena. Inače, sve varijable nađene izvan najdublje domene su nepromjenjive (pokušaj promjene takve varijable će rezultirati izradom *nove* varijable u najdubljoj domeni, ostavljajući identično ime vani nepromijenjeno.

3.3 Razredi

Razredi uvode malo nove sintakse, tri nova tipa objekta i nešto nove semantike.

3.3.1 Definicija razreda

Sintaksa definicija razreda izgleda ovako:

```
class ImeRazreda:
        <naredba-1>
        .
        .
        .
        <naredba-N>
```

Definicija razreda, kao i definicija funkcija (*def* naredba) mora biti izvršena prije nego što ima

U praksi, naredbe unutar razreda su obično definicije funkcija, ali dopuštene su i ostale naredbe koje ponekad mogu biti korisne — o tome će biti riječ kasnije. Definicije funkcija unutar razreda imaju pomalo čudnu listu argumenata, zadanu pozivnim konvencijama za funkcije — također objašnjeno nešto kasnije.

Kad se definira novi razred, također se stvori novi imenik i koristi se u lokalnoj domeni — sva dodjeljivanja lokalnim varijablama odnose se na novi imenik. Konkretno, novonastala funkcija svojom definicijom pohranjuje svoje ime u imenik.

3.3.2 Objekti razreda

Objekti razreda podržavaju dvije vrste operacija: referenciranje atributa i instanciranje.

Referencijranje atributa koristi standardnu sintaksu za sve referenciranje atribuda u Pythonu: `obj.ime`. Valjana imena atributa su sva imena koja se nalaze u u imeniku razreda kada se objekt razreda napravi. S toga, ako definicija razreda izgleda ovako:

```
class MojRazred:
    """Jednostavan primjer razreda"""
    i = 12345
    def f(self):
        return 'hej'
```

tada su `MojRazred.i` i `MojRazred.f` valjane reference atributa, koje vraćaju cjelobrojni podatak (eng. integer) i objekt funkcije, respektivno. Atributi razreda također se mogu dodijeliti kako bi im se vrijednost mogla mijenjati. `__doc__` je također atribut, vračajući znakovni niz koji pripada tom razredu: `"Jednostavan primjer razreda"`.

Instanciranje razreda koristi funkcijsku notaciju. Ako zamislimo da je objekt razreda funkcija bez parametara, a vraća novu instancu razreda. Npr., uzimajući gornji razered:

Instanciranje razreda koristi funkcijsku notaciju. Zamislimo da je objekt razreda funkcija bez argumenata koja vraća novu instancu razreda. Primjer gornjeg razreda:

```
>>> x = MojRazred()
```

stvara novu *instancu* razreda, a taj objekt dodijeljen je lokalnoj varijabli `x`.

Operacija instanciranja (“pozivanje” objekta razreda) stvara prazni objekt. Mnogi razredi rade na taj način, pri instanciranju naprave objekt koji se postavi na neko inicijalno stanje. Zbog toga razred može definirati specijalnu metodu nazvanu `__init__()`, a definira se ovako:

```
def __init__(self):
    self.data = []
```

Kada razred definira `__init__()` metodu, instanciranje razreda automatski poziva `__init__()` za novostvoreni objekt razreda. Za ovaj primjer, nova, inicijalizirana instanca se može napraviti ovako:

```
>>> x = MojRazred()
```

Naravno, `__init__()` metoda može imati argumente, što pridonosi većoj fleksibilnosti. U tom slučaju, argumenti dani razrednom operatoru instanciranja su proslijeđeni `__init__()`-u. Primjer:

```
>>> class KompleksniBroj:
...     def __init__(self, realniDio, imagDio):
...         self.r = realniDio
...         self.i = imagDio
...
>>> x = KompleksniBroj(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

3.3.3 Instance objekta

Operacija koju možemo provoditi na instanci objekta je referenciranje atributa. Poznajemo dvije vrste atributa: podatkovni i metode.

Podatkovni atribut je ono što znamo kao podatkovni član u C++-u. Podatkovni atribut je potrebno deklarirati, poput lokalnih varijabli, te postoje čim im dodijelimo neku vrijednost. Npr., `x` je instanca `MojRazred` razreda. Slijedeći kôd ispisati `16`, a na kraju će atribut biti izbrisan:

```
x.brojac = 1
while x.brojac < 10:
    x.brojac = x.brojac * 2
print x.brojac
del x.brojac
```

Druga vrsta atributa je *metoda*. Metoda je funkcija koja pripada objektu. U Pythonu metoda nije vezana isključivo za instancu razreda jer postoje tipovi objekata koji mogu imati svoje metode. Npr., objekt liste ima metode append, insert, remove, sort, itd.

3.3.4 Metode objekta

Metoda je pozvana odmah nakon što je napravljena poveznica:

```
>>> x.f()
```

U primjeru razreda `MojRazred` rezultat će biti ispis na ekranu `'hej'`. Metoda ne mora biti pozvana odmah: `x.f` je objekt metoda i može biti spremljena za kasniji poziv. Npr:

```
xf = x.f
while True:
    print xf()
```

će ispisivati 'hej' beskonačno puta.

Ako primijetimo x.f() je pozvan bez argumenata, iako je u definiciji funkcije f() specificiran argument. Kod metoda se poziv odvija malo drugačije. x.f() je isti poziv kao i MojRazred.f(x), što znači da metoda prosljeđuje objekt kao prvi argument.

Objekt metoda se napravi tako što se upakira instanca objekta i funkcija objekta. Kada je objekt metoda pozvana s listom argumenata, nova lista se napravi od instance objekta i liste, te je objekt metoda pozvana s novim argumentima.

3.3.5 Napomene u vezi razreda

Podatkovni atributi imaju prvenstvo ispred metoda ako dijele isto ime, da bi se izbjegla kolizija s imenima što vodi do greški koje nije lako naći u velikim programima, predlaže se korištenje neke vrste konvencije koja će smanjiti šansu takvih grešaka. Neki od prijedloga su korištenje prefiksa podatkovnih atributa ili korištenje glagola za metodu, a imenica za podatkovne članove.

Podatkovni atributi mogu se referencirati pomoću metoda kao i direktno. Python nema mehanizam skrivanja podataka. Pythonova implementacija, pisana u programskom jeziku C, je potpuno skrivena i kontrolira pristup određenim objektima.

često je prvi argument metode nazvan self ali to je ništa drugo nego konvencija. Ime self nema nikakvo posebno značenje u Pythonu. No ako ne bi slijedili konvenciju takvog programiranja može samo doprinijeti nečitljivosti kôda a često i programi za pisanje programskog kôda imaju ugrađeno dopunjavanje riječi koje slijede tu konvenciju.

Funkcijski objekt koji je definiran unutar atributa razreda definira metodu tog razreda. No, nije potrebno da definicija funkcije bude unutar definicije razreda. Moguće je dodijeliti funkcijski objekt lokalnoj varijabli razreda. Primjer:

```
# Funkcija definirana izvan razreda
def f1(self, x, y):
    return min(x, x+y)

class C:
    f = f1
    def g(self):
```

```
        return 'pozdrav'
    h = g
```

`f`, `g` i `h` su atributi razreda `C` koji referira funkcijski objekt, i svi su metode `C` – `h` je ekvivalent `g`-a. Takvom praksom inače se ne preporuča programirati jer može uvesti dodatnu nečitljivost kôda.

Metode mogu pozivati druge metode koristeći `self` argument:

```
class B:
    def __init__(self):
        self.data = []
    def dodaj(self, x):
        self.data.append(x)
    def dodajDva(self, x):
        self.dodaj(x)
        self.dodaj(x)
```

Metode mogu referencirati globalna imena isto kao i obične funkcije. Iako rijetko postoji dobar razlog za korištenje globalnih podataka u metodi, postoje razne legitimne upotrebe globalne domene. Funkcije i moduli učitani u globalnu domenu mogu se koristiti u metodi, funkcijama i razredima.

Svaka vrijednost je objekt, pa tako ima i *razred* (također zovemo *tip*). Ta informacija je spremljena u `objekt.__class__`.

3.4 Nasljeđivanje

Pythonu se deklarira kao objektno orijentiran jezik pa s toga podržava i nasljeđivanje. Sintaksa za izvedeni razred izgleda ovako:

```
class IzvedeniRazred(BazniRazred):
        <naredba-1>
        .
        .
        .
        <naredba-N>
```

Ime `BazniRazred` mora biti definiran u istoj domeni kao i izvedeni razred, no moguće je i iz drugih modula naslijediti razred:

```
class IzvedeniRazred(modul.BazniRazred):
```

Pri traženju atributa unutar izvedenog razreda, ukoliko on nije nađen, potraga se odvija unutar baznog razreda. Pravilo se primjenjuje rekurzivno ako i bazni razred je izveden iz nekog razreda.

Metoda izvedenog razreda može imati prvenstvo ispred baznog razreda. Metode nemaju posebne privilegije pri pozivu drugih metoda istog objekta, metoda baznog razreda koja poziva metodu definiranu u istom baznom razredu može završiti pozivom izvedenog razreda koji ima prvenstvo nad njom.

Ako želimo proširiti metodu izvedenog razreda, a ne zamijeniti ju drugom metodom istog imena, postoji jednostavan način poziva metode direktno `BazniRazred.metoda(self, argument)`.

Python ima dvije ugrađene funkcije koje su korisne pri nasljeđivanju:

- `isinstance()` se koristi pri provjeri tipu instance: `isinstance(obj, int)` rezultat će biti `True` ako je `obj.__class__` jednak `int` ili nekom razredu izvedenom iz `int`.
- `issubclass()` se koristi da bi se provjerilo razredovo nasljeđe: `issubclass(bool, int)` je `True` ako `bool` je podrazred `int`-a. No, `issubclass(unicode, str)` je `False` jer `unicode` nije podrazred od `str`

3.4.1 Višestruko nasljeđivanje

Python podržava i višestruko nasljeđivanje. Definicija razreda koji koristi višestruko nasljeđivanje izgleda ovako:

```
class IzvedeniRazred(Baza1, Baza2, Baza3):
        <statement-1>
        .
        .
        .
        <statement-N>
```

Pravilo koje vrijedi je "prvo dubina" i "s lijeva na desno". Stoga, ako atribut nije nađen u `IzvedeniRazred`, potraga se nastavlja u `Baza1`, te zatim rekurzivno u bazi razreda `Baza1`, ako ni tamo nije nađen pretražuje se `Baza2` itd.

3.5 Privatne varijable

"Privatne" instance varijabli kojima se ne može pristupiti osim unutar objekta ne postoje u Python jeziku. No postoji konvencija kojom se koriste Python programeri: ime s prefiksom povlake (npr. `_ime`) trebalo bi se tretirati kao nejavni dio programskog kôda bilo da se radi o podatkovnom članu ili metodi, te bi se trebalo smatrati takav atribut implementacijom.

Kako postoji potreba za postojanje privatnih članova klase (najprije da se izbjegne kolizija imena s imenima definiranim u podrazredu), postoji ograničeni mehanizam nazvan *name mangling*. `__ime` (dvije povlake) se interno zamjenjuje s `_razred__ime`, gdje je `razred` je trenutni razreda s početnom povlakom.

Promjena imena je korisna pri korištenju podrazreda:

```
class mapa:
    def __init__(self, lista):
        self.lista_elemnata = []
        self.__azuriraj(lista)

    def azuriraj(self, lista):
        for i in lista:
            self.lista_elemnata.append(i)

    __azuriraj = azuriraj   # privatna kopija originalne metode

class mapa_podrazred(mapa):

    def azuriraj(self, kljuc, vrijednost):
        for i in zip(kljuc, vrijednost):
            self.lista_elemnata.append(i)
```

Ovakav mehanizam je dizajniran iz razloga se ne dogode slučajne pogreške, još uvijek je članovima moguće pristupiti ili promijeniti im vrijednost iako se smatraju privatnima.

3.6 Iznimke

Iznimke definirane od strane korisnika se također identificiraju kao razredi.

Primjer ključne riječi *raise*:

```
raise Class, instance
```

```
raise instance
```

U prvoj formi `instance` mora biti instanca `Class` ili razred izveden iz njega. Druga je skraćeni zapis:

```
raise instance.__class__, instance
```

Razered u *except* naredbi je kompatibilan s iznimkom ako je istog razreda ili izvedenica baznog razreda (ali ne i obrnuto). U primjeru ispis će biti B, C, D:

```
class B:
    pass
class C(B):
    pass
class D(C):
    pass

for c in [B, C, D]:
    try:
        raise c()
    except D:
        print "D"
    except C:
        print "C"
    except B:
        print "B"
```

Ako bi if uvjeti bili obrnutim redoslijedom (`except B` prvi), ispis bi bio B, B, B — prva iznimka koja ispunjava uvjet je pokrenuta.

Kada se ispiše poruka greške za nepredviđenu iznimku, ispiše se ime razreda, mjesto greške, te instanca pretvorena u znakovni niz uz pomoć ugrađene funkcije `str()`.

3.7 Iteratori

Kao što je poznato kontejneri objekata mogu se prebrojavati pomoću ključne riječi *for*:

```
for element in [1, 2, 3]:
    print element
for element in (1, 2, 3):
```

```
    print element
for kljuc in {'jedan':1, 'dva':2}:
    print kljuc
for znak in "123":
    print znak
for linija in open("moj.txt"):
    print linija
```

Princip je jednostavan i jasan. Korištenje iteratora prožet je kroz Python. Ono što se dešava u pozadini je da *for* poziva `iter()` funkciju nad kontejnerom objekata. Funkcija vraća iterator objekta koji definira metodu `next()` koji pristupa kontejneru i svakom objektu pojedinačno. Kada u kontejneru više nema elemenata, `next()` poziva iznimku `StopIteration` koji prekida *for* petlju.

```
>>> s = 'abc'
>>> it = iter(s)
>>> it
<iterator object at 0x00A1DB50>
>>> it.next()
'a'
>>> it.next()
'b'
>>> it.next()
'c'
>>> it.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
    it.next()
StopIteration
```

Kako je poznata unutarnja mehanika protokola iteratora, lako je dodati iterator vlastitom razredu. Prvi korak je definiranje `__iter__()` metode kojoj je povratna vrijednost objekt s metodom `next()`. Ako razred definira `next()` metodu, `__iter__()` može jednostavno vratiti `self`:

```
class Obrni_niz:
    """Iterator za ispisivanje riječi unatrag."""
    def __init__(self, data):
        self.data = data
        self.index = len(data)
    def __iter__(self):
        return self
```

```
    def next(self):
        if self.index == 0:
            raise StopIteration
        self.index = self.index - 1
        return self.data[self.index]

>>> niz = Obrni_niz('spam')
>>> iter(niz)
<__main__.Obrni_niz object at 0x00A1DB50>
>>> for znak in niz:
...     print znak
...
m
a
p
s
```

3.8 Generatori

Generatori su jednostavan i moćan alat za izradu iteratora. Napisani su kao obične funkcije ali koriste ključnu riječ *yield* kad vraća vrijednost. Svaki put kada je `next()` pozvan, generator nastavlja gdje je stao (pamti sve podatke i koja je naredba zadnja izvršena). Primjer pokazuje jednostavan primjer generatora:

```
def obrni(niz):
    for indeks in range(len(niz)-1, -1, -1):
        yield niz[indeks]

>>> for char in obrni('golf'):
...     print char
...
f
l
o
g
```

Sve što se može napraviti s generatorima može se i uz pomoć razreda s iteratorom kao što je opisano. Ono što generatore čini kompaktnima je da `__iter__()` i `next()` se naprave automatski.

Lokalne varijable i stanje odvijanja programa automatski su spremljeni između poziva. To

čini funkciju puno lakšom za napisati i jasnijom za shvatiti nego pristup korištenjem instanci varijabli kao što su `self.indeks` i `self.niz`.

Automatsko stvaranje metode i spremanje stanja programa, generatori i automatski pozivaju iznimku `StopIteration` kada generator završi. Sve te značajke u kombinaciji čine iteratore jednostavnim za pisanje i korištenje.

3.8.1 Izrazi generatora

Primjer jednostavnog korištenja generatora koristeći sintaksu sličnu listama ali sa oblim zagradama umjesto uglatima. Takvi izrazi su osmišljeni za situacije gdje se generatori koriste odmah u sprezi s nekim drugim stvarima. Izrazi s generatorima su više kompaktni ali manje svestrani od definiranja potpunog generatora, a i utrošak memorije je manji.

Primjer:

```
>>> sum(i*i for i in range(10))                 # zbroj kvadrata
285

>>> xvek = [10, 20, 30]
>>> yvek = [7, 5, 3]
>>> sum(x*y for x,y in zip(xvek, yvek))         # skalarni produkt
260

>>> from math import pi, sin
>>> sine_table = dict((x, sin(x*pi/180)) for x in range(0, 91))

>>> niz = 'golf'
>>> list(niz[i] for i in range(len(niz)-1,-1,-1))
['f', 'l', 'o', 'g']
```

MOMENTI

Brojni problemi u mehanici, fizici i inženjerstvu vode do problema karakterizacije neke funkcije u obliku nekog funkcionala. Funkcional je funkcija koja iz vektorskog prostora pretvara u skalarno polje, drugim riječima funkcional kao argument uzima vektor, a vraća skalar. Momenti su skalarne kvantitete koje koristimo kako bi karakterizirali funkciju i pri tome dobili njene jedinstvene značajke. Momenti se opsežno koriste u statistici za opisivanje oblika vjerojatnosti gustoće funkcije (eng. probability density function), a u mehanici krutih tijela koriste se kao mjera distribucije mase. S matematičkog gledišta, momenti su projekcija funkcije na polinomnu bazu (slično kako je Fourierova transformacija projekcija na bazu harmoničke funkcije).

Sliku možemo promatrati kao funkciju $f(x, y)$ s dvije varijable koja ima konačni integral koji nije jednak nuli. Opći moment $m_{pq}(f)$ slike $f(x, y)$, gdje p, q su pozitivni cjelobrojni brojevi, a r = p + q se naziva redom momenta koji je definiran ovako:

$$m_{pq} = \iint_D x^p y^q f(x, y) dx dy \tag{4.1}$$

gdje su $p_{00}(x, y), p_{10}(x, y), ..., p_{kj}(x, y), ...$ polinomne baze funkcije. Ovisno o vrsti korištenog polinoma, dobivamo razne tipove momenata.

4.1 Geometrijski i kompleksni momenti

Najčešći izbor baze je klasična potencijska baza $p_{kj}(x, y) = x_k y_j$ što daje geometrijske momente:

$$m_{pq} = \int_{\infty}^{\infty} \int_{\infty}^{\infty} x^p y^q f(x, y) dx dy \tag{4.2}$$

Geometrijski momenti niskog reda imaju svojevrsno značenje pa će biti pobrojani prema redu momenta. m_{00} odnosno moment nultog reda predstavlja „masu“ slike (za binarne slike odnosno slike koje imaju samo crne i bijele piksele), tj. masu dane funkcije slike $f(x, y)$.

$$m_{00} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} f(x, y) dx dy \tag{4.3}$$

Momenti prvog reda

$$m_{10} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} x f(x, y) dx dy \tag{4.4}$$

i

$$m_{01} = \int_{a_1}^{a_2} \int_{b_1}^{b_2} y f(x, y) dx dy \tag{4.5}$$

predstavljaju centar mase slike $f(x, y)$. Centar mase slike je točka u kojoj bi mogla biti koncentrirana cijela masa slike bez da se promjeni moment slike oko bilo koje osi. U 2D slučaju, koordinate centra mase su

$$\bar{x} = \frac{m_{10}}{m_{00}}, \bar{y} = \frac{m_{01}}{m_{00}} \tag{4.6}$$

m_{10}/m_{00} i m_{01}/m_{00} definiraju centar gravitacije slike odnosno njeno težište. Momenti drugog reda m_{20}, m_{02}, m_{11} opisuju distribuciju mase slike s obzirom na koordinatne osi. U mehanici poznato pod imenom inercijski moment, a mogu se koristiti za određivanje bitne značajke slike - orijentacije. Slijedeće svojstvo koje možemo povezati s mehanikom krutog tijela, izraženo preko momenata je radijus okretanja $\sqrt{m_{20}/m_{00}}$ i $\sqrt{m_{02}/m_{00}}$. U mehanici krutog tijela izražavamo ga kao odnos inercijskog momenta i mase.

Alternativni način da se opišu značajke slike pomoću momenata je da se razmotri odnos između momenata slike i njihovih projekcija na sliku. Momenti u setu m_{p0} i m_{0p} su ekvivalentni momentima projekcije slike na x osi i y osi respektivno. Razmotrimo horizontalnu projekciju,

h(y), slike f(x,y) na y os danu jednadžbom

$$h(y) = \int_{a_1}^{a_2} f(x,y)dx. \tag{4.7}$$

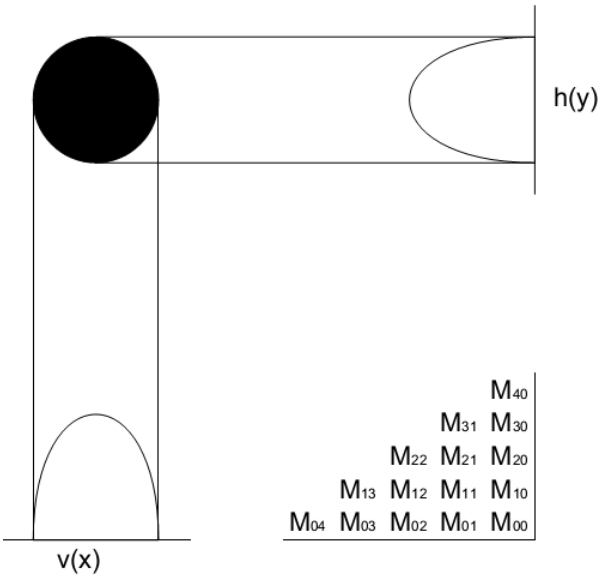
Tada jednodimenzijski moment, m_q, od h(y) su dobiveni

$$m_q = \int_{b_1}^{b_2} y^q h(y)dy. \tag{4.8}$$

Uvrštavanjem (4.7) u (4.8) daje

$$m_q = \int_{a_1}^{a_2}\int_{b_1}^{b_2} y^q f(x,y)dxdy = m_{0q} \tag{4.9}$$

Slika 4.1 prikazuje projekciju objekta na x i y os i podskupinu momenata odgovornih za projekciju.



Slika 4.1: Projekcija momenata na x i y os.

Drugi popularan izbor polinomne baze $p_{kj}(x,y) = (x+iy)^k + (x-iy)^j$, gdje je i imaginarna

jedinica, što daje tzv. kompleksne momente

$$c_{pq} = \int_{\infty}^{\infty}\int_{\infty}^{\infty}(x+iy)^p + (x-iy)^q f(x,y)dxdy \tag{4.10}$$

Samo indeksi $p \geq q$ su značajni jer $c_{pq} = c_{qp}^*$. Zvjezdica označava da je konjugirano kompleksan broj. Geometrijski i kompleksni momenti nose jednaku količinu informacija. Svaki kompleksni moment moguće je izraziti preko geometrijskog momenta istog reda:

$$c_{pq} = \sum_{k=0}^{p}\sum_{j=0}^{q}\begin{pmatrix} p \\ k \end{pmatrix}\begin{pmatrix} q \\ j \end{pmatrix}(-1)^{q-j} \times i^{p+q-k-j} \times m_{k+j,p+q-k-j} \tag{4.11}$$

i obrnuto:

$$m_{pq} = \frac{1}{2^{p+q}i^q}\sum_{k=0}^{p}\sum_{j=0}^{q}\begin{pmatrix} p \\ k \end{pmatrix}\begin{pmatrix} q \\ j \end{pmatrix}(-1)^{q-j} \times c_{k+j,p+q-k-j} \tag{4.12}$$

Kompleksni momenti su uvedeni zbog njihove pogodne osobine pri rotaciji slike. Te osobine mogu biti iskorištene pri konstrukciji invarijanti s obzirom na rotaciju.

4.2 Invarijantni momenti na translaciju, rotaciju i skaliranje

Translacija, rotacija i skaliranje su najjednostavnije manipulacije koje možemo izvesti nad slikom. Te transformacije možemo izraziti pomoću četiri parametra:

$$\mathbf{x}' = s\mathbf{R} \times \mathbf{x} + \mathbf{t}, \tag{4.13}$$

gdje je t vektor translacije, s je faktor skaliranja (govorimo o uniformnom skaliranju gdje se po horizontali i vertikali jednako smanjuje ili proširuje slika), dok je R matrica rotacije izražena

kao

$$\mathbf{R} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \tag{4.14}$$

gdje je α kut rotacije.

Invarijante na translaciju, rotaciju i skaliranje su korisne u velikom broju slučajeva jer objekt treba prepoznati bez obzira na orijentaciju i poziciju objekta u sceni te udaljenosti od uređaja za uzimanje slike.

4.3 Invarijanta na translaciju

Invarijanta na translaciju može se jednostavno izvesti tako što se objekt naizgled premjesti tako da se težište podudara s ishodištem koordinatnog sustava, i obrnuto, premještanjem polinomne baze u težište objekta. U slučaju geometrijskih momenata, postoje tzv. centralni geometrijski momenti

$$\mu_{pq} = \int_{\infty}^{\infty}\int_{\infty}^{\infty}(x+x_c)^p + (y-y_c)^q f(x,y)dxdy, \tag{4.15}$$

gdje su

$$x_c = \frac{m_{10}}{m_{00}}, y_c = \frac{m_{01}}{m_{00}} \tag{4.16}$$

koordinate težišta objekta. Uvijek će biti μ10 = μ10 = 0 i μ00 = m00. Centralne momente moguće je izraziti pomoću geometrijskih kao

$$\mu_{pq} = \sum_{k=0}^{p}\sum_{j=0}^{q}\begin{pmatrix} p \\ k \end{pmatrix}\begin{pmatrix} q \\ j \end{pmatrix}(-1)^{k+j} \times x_c^k \times y_c^j \times m_{p-k,q-j} \tag{4.17}$$

Iako njihova povezanost nema veliki značaj za teorijsko razmatranje, ponekad se koristi za izračun centralnih momenata pomoću brzog algoritma za izračun geometrijskih momenata.

4.4 Invarijanta na uniformno skaliranje

Invarijanta na uniformno skaliranje se dobije normalizacijom svakog momenta. U principu, svaki moment se može koristiti kao normalizacijski faktor ako je različit od nule za sve slike. Kako momenti nižeg reda imaju veću stabilnost na šum i lakši su za izračun, najčešći je izbor μ00 s određenom potencijom

$$\nu_{pq} = \frac{\mu_{pq}}{\nu_{00}^{\omega}} \tag{4.18}$$

gdje je

$$\omega = \frac{p+q}{2} + 1 \tag{4.19}$$

Moment ν_{pq} se zove normalizirani centralni geometrijski moment. Nakon skaliranja faktorom s, centralni momenti u novim koordinatama će izgledati ovako:

$$\mu_{pq} = \int_{\infty}^{\infty}\int_{\infty}^{\infty}(x^{'} + x_c^{'})^p + (y^{'} - y_c^{'})^q f(x^{'}, y^{'})dx^{'}dy^{'}. \tag{4.20}$$

Odnosno

$$\mu_{00}^{'} = s^2\mu_{00} \tag{4.21}$$

Čime je

$$\nu_{pq}^{'} = \frac{\mu_{pq}^{'}}{\left(\mu_{00}^{'}\right)^{\omega}} = \frac{s^{p+q+2}\mu_{pq}}{\left(s^2\mu_{00}\right)^{\omega}} = \nu_{pq} \tag{4.22}$$

što dokazuje da je normalizirani moment invarijantan na skaliranje. Moment koji je korišten za normalizaciju više se ne može koristiti za prepoznavanje jer vrijednost zbog normalizacije će biti uvijek jednaka jedan. Ako želimo zadržati nulti red momenta valjanim, možemo normalizirati drugim momentom ali obično se normalizira nultim redom, iznimke su rijetke. Jedina značajna druga normalizacija je $(\mu_{20} + \mu_{02})^{\omega/2}$

4.5 Prve invarijante na rotaciju

Najraniji značajan rad u kojem su momenti korišteni u svrhu procesiranje slike i prepoznavanja uzoraka napravio je Hu. Na temelju teorije algebre invarijanti, Hu je izveo kombinacije momenata koje su nepromjenjive s obzirom na rotaciju oko ishodišta

$$\phi_1 = m_{20} + m_{02}, \tag{4.23}$$

$$\phi_2 = (m_{20} - m_{02})^2 + 4m_{11}^2, \tag{4.24}$$

$$\phi_3 = (m_{30} - 3m_{12})^2 + (3m_{21} - m_{03})^2, \tag{4.25}$$

$$\phi_4 = (m_{30} - m_{12})^2 + (m_{21} - m_{03})^2, \tag{4.26}$$

$$\begin{aligned}\phi_5 = (m_{30} - 3m_{12})(m_{30} + m_{12})((m_{30} + m_{12})^2 - 3(m_{21} + m_{03})^2) \\ +(3m_{21} - m03)(m_{21} + m_{03})(3(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2)\end{aligned} \tag{4.27}$$

$$\phi_6 = (m_{20} - m_{02})((m_{30} + m_{12})^2 - (m_{21} + m_{03})^2 + 4m_{11}(m_{30} + m_{12})(m_{21} + m_{03}), \tag{4.28}$$

$$\begin{aligned}\phi_7 = (3m_{21} - m_{03})(m_{30} + 3m_{12})((m_{30} + m_{12})^2 - 3(m_{21} + m_{03})^2) \\ -(m_{30} - 3m_{12})(m_{21} + m_{03})(3(m_{30} + m_{12})^2 - (m_{21} + m_{03})^2).\end{aligned} \tag{4.29}$$

Ako geometrijske momente zamijenimo centralnim ili normaliziranim momentima dobiti ćemo invarijante ne samo na rotaciju već i na translaciju i skaliranje. Iako Hu-ove invarijante imaju problem zbog ograničene moći prepoznavanja, međusobnih ovisnosti i ograničenja na momente do trećeg reda, imaju veliku primjenu u mnogim područjima. Slaba točka Hu-ove teorije je što ne pruža mogućnost generalizacije. Koristeći se njome, nije moguće izvesti invarijante iz višeg reda momenta i invarijante općenitijim transformacijama.

4.6 Konstrukcija baze

Možemo napraviti beskonačan broj invarijanti za bilo koji red momenta ali samo nekoliko ih je međusobno neovisna. Pod pojmom baza misli se na najmanji set potreban da bi se izrazilo ostale invarijante. Točnije baza mora biti neovisna što znači da ni jedan od njenih elemenata se ne može izraziti kao funkcija drugih elemenata, također i konačna odnosno da rotacijske invarijante se mogu izraziti samo uz pomoć elemenata baze. Poznavanje baze je ključno jer u zadacima prepoznavanja uzoraka baza pruža jednaku diskriminaciju kao set svih invarijanti,a tako smanjujemo vrijeme potrebno za računanje invarijanti. Na primjeru

$\{c_{20}c_{02}, c_{12}^2c_{20}, c_{21}c_{12}, c_{21}^3c_{02}c_{12}\}$

je ovisan set čija baza je $\{c_{12}^2c_{20}, c_{21}c_{12}\}$.

Da bi formalizirali ove pojmove, uvodimo slijedeće definicije.

Definicija: Neka je $k \geq 1$ i $I = \{I_1, ..., I_k\}$ je set rotacijskih invarijanti. J je rotacijska invarijanta. Kažemo da je J ovisan o I samo ako postoji funkcija F od k varijabli takva da je $J = F(I_1, ..., I_k)$. Inače je I neovisan.

Definicija: Neka je $k > 1$ i $I = \{I_1, ..., I_k\}$ set rotacijskih invarijanti. I je ovisan samo ako postoji $k_0 \leq k$ takav da je I_{k0} ovisan o $I - \{I_{k0}\}$. Inače je I neovisan.

$\{c_{20}c_{02}, c_{12}^2c_{20}, c_{21}c_{12}, c_{21}^3c_{02}c_{12}\}$ i $\{c_{20}^2c_{12}^2, c_{02}c_{21}^2\}$ je primjer ovisnog seta invarijanti.

Definicija: Neka je I set rotacijskih invarijanti i B njihov podniz. B je zvan potpunim podnizom samo ako bilo koji element I - B ovisan o B. Set B je baza za I samo ako je neovisan i potpun.

Sada možemo formulirati temeljni teorem koji govori kako konstruirati bazu invarijanti određenog reda.

Teorem: Razmotrimo kompleksne momente za red $r \geq 2$. Rotacijske invarijante B konstruiramo kako slijedi:

$$\mathcal{B} = \{\Phi(p,q) \equiv c_{pq} c_{p0pq}^{p-q} | p \geq q \bigwedge p + q \leq r\}, \tag{4.30}$$

Gdje su p_0 i q_0 proizvodljni indeksi tako da $p_0 + q_0 \leq r, p_0 - q_0 = 1$ i $c_{p0q0} \neq 0$. Tada je B baza za sve rotacijske invarijante momenata bilo koje vrste do reda r.

Baza definirana teoremom općenito nije jedinstvena. Ona ovisi o izboru p_0 i q_0 koji su vrlo bitni. Prirodno, nameče se pitanje kako izabrati te indekse u praksi. S jedne strane, želimo indekse p_0 i q_0 zadržati što nižima jer su momenti nižeg reda manje osjetljivi na šum od onih višeg reda. Dok s druge strane, momenti c_{p0q0} koji imaju vrijednosti blizu nuli mogu uzrokovati numeričku nestabilnost invarijanti. Stoga, predložen je slijedeći algoritam. Počinjemo od $p_0 = 2$ i $q_0 = 1$ te provjerimo prelazili li $|c_{p0q0}|$ definiranu granicu za sve objekte. Ako je uvjet zadovoljen, p_0 i q_0 su odabrani koji jesu, ako nije - povećamo p_0 i q_0 za jedan i ponovimo proceduru.

4.7 Baza invarijanti drugog i trećeg reda

Prema teoremu koji govori kako dobiti bazu zadanog reda $r \geq 2$ dobili smo

$$\theta(1,1) = c_{11}, \tag{4.31}$$

$$\theta(2,1) = c_{21} c_{12}, \tag{4.32}$$

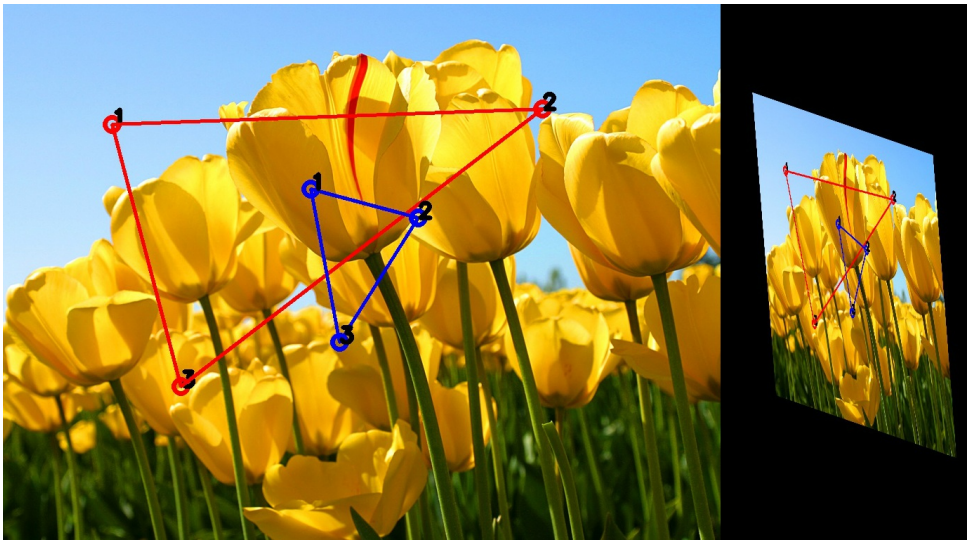
$$\theta(2,0) = c_{20} c_{12}^2, \tag{4.33}$$

$$\theta(3,0) = c_{30} c_{12}^3, \tag{4.34}$$

Baza ima šest komponenti koje predstavljaju invarijante. Prema teoremu baza bi također trebala sadržavati invarijante $\phi(0,0) = c_{00}$ i $\theta(1,0) = c_{10}c_{12}$, no te invarijante nisu uključene u bazu jer $c_{00} = \mu_{00}$ se najčešće koristi kao normalizacija za invarijantu skaliranja, a $c_{10} = m_{10} + im_{01}$ se koristi da bi se postigla invarijanta na translaciju. Stoga $\theta(0,0) = 1$, $\theta(1,0) = 0$ i nepotrebno ih je razmatrati.

4.8 Afine transformacije

Afine transformacije su linearne transformacije tj. transformacije koje očuvaju ravne linije (sve točke koje inicijalno leže na nekoj liniji, i dalje će biti na toj liniji nakon transformacije) i odnose udaljenosti između točki. Transformacije translacije, rotacije i skaliranja su samo posebni slučajevi afinih transformacija.



Slika 4.2: Prikaz afinih transformacija.

Na desnoj strani slike 4.2 prikazana je afina transformacija lijeve slike. Da bi napravili afinu transformaciju slike potrebno nam je šest točki. Prve tri točke formiraju crveni trokut koji služi kao mapa za transformaciju u druge tri točke odnosno plavi trokut.

4.9 Afini invarijantni momenti

Afini momenti otkriveni su prije od prilike 50 godina i nije sigurno tko je objavio prve točne i u praksi primjenjive momente. Prvi invarijantni momenti na afine transformacije predstavio je Hu ali je Temeljni teorem afinih invarijanti bio krivo naveden. Trideset godina kasnije, Reiss i Flusser, Suk su neovisno otkrili i ispravili grešku, objavili novi set momenata i dokazali

njihovu korisnost u jednostavnim zadacima prepoznavanja. Afini momenti se mogu izvesti na nekoliko načina, a razlikuje se u korištenom matematičkom alatu. Algebarskim invarijantama, teorijom grafova, tenzorskom algebrom, direktnim rješavanjem parcijalne diferencijalne jednadžbe i normalizacijom slike mogu se izvesti afini momenti. Momenti izvedeni na bilo koji od navedenih načina su ekvivalentni. Zajedničko svim metodama je da mogu generirati koliko god invarijanti ali samo neke od njih biti će neovisne. Ovisne invarijante su u praksi nekorisne, također je uloženo puno truda u njihovu eliminaciju.

4.10 Moguće zavisnosti između afinih momenata

Identifikacija i eliminacija ovisnih momenata je bitan i izrazito kompliciran zadatak. Moguće su razne vrste ovisnosti između invarijanti. Kategorizirane su u četiri skupine uz opis kako ih eliminirati.

1. Invarijante mogu biti jednake nuli bez obzira o kojoj i kakvoj se slici radi.
2. Ponekad se pojave identične invarijante.
3. Postoje invarijante koje su jednake umnošku drugih invarijanti, eliminacija se vrši istraživanjem svih mogućnosti umnožaka te provjerom njihove neovisnosti.
4. Invarijante mogu biti linearna kombinacija drugih invarijanti ili njihovog produkta.

Nakon eliminacije gornjih ovisnosti dobivamo tzv. nesvodljive invarijante odnosno invarijante koje se ne daju više smanjiti. No, „nesvodljivi“ ne znači i neovisni - moguće su polinomne ovisnosti višeg reda. Ako je takav odnos uočen potrebno je eliminirati sve sudionike invarijanti osim jedne. Ovisnosti 1 i 2 je jednostavno za naći. Za eliminacija umnoška potrebno je dubinski pretraživati invarijante. Sve moguće kombinacije parova invarijanti su pomnožene te se provjeri njihova neovisnost. Da bi se eliminiralo tip 4 moraju se provjeriti sve moguće linearne kombinacije invarijanti. Kako samo invarijante jednake strukture mogu biti linearno ovisne, invarijante se sortiraju prema strukturi (uključujući sve produkte). Za svaku grupu iste strukture formira se matrica koeficijenata svih invarijanti. Rank matrice je jednak broju linearno neovisnih invarijanti date strukture. Na hipotetičkom primjeru prikazan polinomne ovisnosti između invarijanti. Pretpostavimo da su I_1, I_2, I_3, I_4 nesvodive invarijante s tri ovisnosti:

S1:

$$I_1^2 + I_2 I_3 = 0 \tag{4.35}$$

S2:

$$I_4^2 - I_2 I_3^2 = 0 \tag{4.36}$$

S3:

$$I_1^4 + 2I_1^2 I_2 I_3 + I_4^2 I_2 = 0 \tag{4.37}$$

Ako tvrdimo da su tri invarijante neovisne i prijeđemo preko njih, to bi bila greška jer treća invarijanta je kombinacija prve dvije:

$$S_1^2 + I_2 S_2 - S_3 = 0 \tag{4.38}$$

Te time ne donosi novu informaciju. Između ove četiri invarijante, dvije su neovisne i dvije su ovisne.

Popis svih 362 afinih invarijanti nalazi se u [8]. Programski kôd za momente korištene u ovom radu nalaze se u dodatku.

STROJNO UČENJE

5.1 Definicija strojnog učenja

Strojno učenje je grana umjetne inteligencije, znanstvena disciplina koja se bavi razvojem i konstrukcijom algoritama koji kao ulaznu vrijednost uzimaju empirijske podatke, npr. podaci iz baze podataka ili trenutno očitanje senzora, a daju predikacijski mehanizam na temelju generiranih podataka. Glavni fokus strojnog učenja je dizajn algoritma koji može prepoznati kompleksne uzorke i donijeti inteligentnu odluku baziranu na ulaznom podatku. Ti algoritmi često koriste heurističke, statističke i empirijske metode kako bi uspjeli reducirati prostor pretraživanja i riješili probleme koje klasični algoritmi ne uspijevaju.

5.2 Klasifikacija

Metode strojnog učenja najčešće se koriste za probleme klasifikacije. Pretpostavimo da imamo objekt, opisan s mnogo atributa. Svakom objektu se može dodijeliti točno jedna klasa iz konačnog skupa mogućih klasa. Atributi su nezavisno promatrane varijable, diskretne ili kontinuirane. Klasa je zavisna nepromatrana varijabla i njenu vrijednost određujemo iz vrijednosti odgovarajućih nezavisnih varijabli. Klasifikacija je svrstavanje podataka u jednu ili više unaprijed definiranih kategorija.

Podatci s kojima se uče algoritmi sastoje se od primjera koji opisuju riješene primjere danog problema. Različiti klasifikatori reprezentiraju funkciju preslikavanja na različite načine, pa tako imamo: stabla odlučivanja, pravila odlučivanja, naivni Bayesov klasifikatori, Bayesove mreže vjerovanja, klasifikatori najbližih susjeda, linearne diskriminantne funkcije, logička regresija, strojevi s potpornim vektorima i umjetne neuronske mreže.

Klasifikacijski proces se može kategorizirati u dva tipa:

- Klasifikacija s nadziranim učenjem: u ovom tipu klasifikacije postoji netko tko nadzire učenje i prepoznavanje sistema kako klasificirati poznati set uzoraka.

- Klasifikacija bez nadziranog učenja: klasifikacijski proces nije ovisan o prijašnjim informacijama. Klasteriranje je nenadzirana klasifikacija, koji je proces stvaranje klasa bez prijašnjeg znanja o uzorcima.

5.3 k najbliži susjed

U klasifikaciji i prepoznavanju uzoraka, „k – najbliži susjed“ algoritam (k – nearest neighbor algorithm, skraćeno k-NN) je metoda za klasifikaciju objekta na temelju podataka kojima je algoritam naučen odnosno treniran. k-NN je tip algoritma koji temelji svoje znanje na instancama koje ima u memoriji, a te instance su spremljene u memoriju prilikom učenja algoritma. Takvi tipovi algoritma kao što je k-NN ne vrši eksplicitnu generalizaciju podataka već uspoređuje novu instancu s onim predstavljenim kroz trening tj. što je pohranjeno u memoriju. Time zapravo nikad ne izgrađuju općenitu ciljnu funkciju za cijeli prostor primjera – što je zapravo prednost, jer se umjesto jedne ciljne funkcije za cijeli prostor gradi nova, lokalna ciljna funkcija za dio prostora primjera u kojem se nalazi klasificirani primjer. To je svojstvo naročito korisno kada je ciljna funkcija prekompleksna za definiranje nad čitavim prostorom, ali se ipak može prikazati skupom lokalnih aproksimacija, u dijelovima prostora primjera. Takve vrste algoritma spadaju u kategoriju lijenih algoritama. To znači da se generalizacija naučenih podataka odgađa do novog upita. k-NN je najjednostavniji algoritam za strojno učenje: objekt je klasificiran prema većinskom glasu njegovih susjeda. Objekt je dodijeljen klasi koja ima najveći broj ponavljanja u k najbližih susjeda (k je pozitivan cijeli broj).

Podaci za treniranje algoritma mogu biti višedimenzionalni vektori s oznakom klase kojoj pripadaju. Treniranje se sastoji od spremanja vektora i njihovih oznaka. U klasifikacijskoj fazi, k je korisnički definirana konstanta, neimenovani vektor je klasificiran tako što mu je dodijeljeno ime klase kojoj pripada prema najvećem broju istoimenih k trening podataka. Negativna strana klasifikacije glasanjem većine je što klasa s većim brojem uzoraka ima tendenciju dominiranja u predikciji novih vektora, jer će njihova zastupljenost biti veća u k najbližih susjeda. Jedan od načina kako riješiti taj problem je dodati težine na udaljenosti između točki.

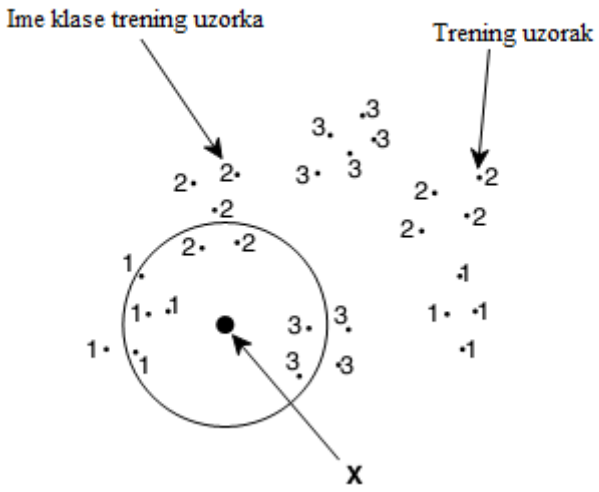
Najbolji odabir broja k ovisi o podacima. Generalno, velika vrijednost k smanjuje efekt šuma u klasifikaciji, ali tada granice između klasa postaju manje izražene. Dobra vrijednost k može se odabrati raznim heurističkim tehnikama, npr. križnom validacijom. Križna validacija (eng. cross-validation) je tehnika za procjenjivanje kako će rezultat statističke analize generalizirati se na nezavisnom setu podataka. Uglavnom se koristi tamo gdje je cilj predikcija, i želi se estimirati točnost prediktivnog modela u praksi. Jedan ciklus križne validacije se sastoji od particioniranja podataka u komplementarne podskupine, provodeći analizu na jednoj podskupini (zvana trening skupina) i validaciju analize na drugoj skupini (zvana validacijska skupina ili testna skupina). Da bi se smanjila varijabilnost, provodi se više ciklusa križne validacije i

izvodi se korištenjem različitih dijelova. Rezultat validacije se dobije prosjekom svih ciklusa.

Točnost k-NN algoritma može se značajno smanjiti prisutnošću šumovitih i nepotrebnih značajki.

Princip koji stoji iza metode najbližeg susjeda je da se nađe predefinirani broj k trening uzoraka najbližih po udaljenosti između njih i nove točke, te da se predvidi njena oznaka iz njih. Udaljenost između dvije točke definira se kao Euklidska udaljenost.

Glavna prednost lijenih metoda učenja je što se ciljna funkcija aproksimira lokalno, kao što je slučaj u k-NN algoritmu. Zato što se ciljna funkcija aproksimira lokalno za svaki upit pretrage, lijeni sustavi mogu istovremeno rješavati više problema, pa čak da i dolazi do promjene u domeni problema. Negativna strana takvih metoda je što zahtijevaju veliki prostor za pohranu podataka učenja. Značajno zašumljeni podaci za učenje samo povećavaju bazu nepotrebno, jer ne uvode novu apstrakciju u fazi učenja. Još jedan nedostatak je brzina. Lijene metode su obično sporiji u evaluaciji, iako kompenziraju s bržom fazom učenja. Lijene metode su pogodne za velike setove podataka s malo atributa (male dimenzije vektora).



Slika 5.1: Primjer klasifikacije k-NN algoritmom

X je uzorak koji je potrebno klasificirati. Ako uzmemo da je k = 8, četiri uzorka pripada klasi 1, dva uzorka su iz klase 2 i dva iz klase 3. Ispitni uzorak klasificiran je kao klasa 1 jer je većina uzoraka kojima je okružen iz klase 1.

Pozivanje k-NN algoritma iz sklearn modula moguće je na sljedeći način:

```
from sklearn import neighbors
data_set = [[1,1],[0,1],[1,0],[3,3],[3.1,3],[2.9,3]]
klase = [1,1,1,2,2,2]
k = 3
clf = neighbors.KNeighborsClassifier(k, weights='distance')
clf.fit(data_set, lable)
rez = clf.predict([1.5, 0])
```

U navedenom programskom kôdu zadan je set podataka s kojim se algoritam uči. Učenje se odvija u liniji kôda gdje pozivamo funkciju *fit*. Parametre algoritma namještamo pri stvaranju objekta *clf*, a to su broj *k*, te kako će se računati udaljenost između točki. Pomoću funkcije *predict* dobijemo rezultat algoritma za traženi upit.

5.4 Križna validacija parametara k-NNa

Dio sklearn-ovog modula uključuje i funkcije za križnu validaciju i provjeru performansi, a nalaze se u sklearn.cross_validation. Analizu je potrebno provesti kako bi znali optimalne parametre algoritma te učinkovitost algoritma s određenim parametrima.

Pomoću križne validacije možemo provjeriti stanje za određenu konfiguraciju, a to znači da ako prikažemo na grafu invarijantu 1 i 2, nećemo dobiti istu raspodjelu kao 1 i 10. Stoga, moramo pronaći i konfiguraciju koja daje najbolje rezultate, a ne samo parametre algoritama.

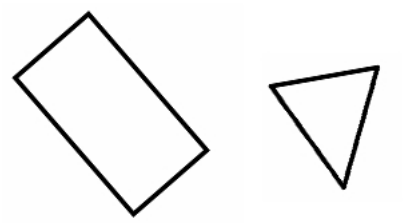
Također, križnu validaciju koristimo ako nemamo velik set podataka, a želimo ga iskoristiti za učenje i provjeru učenja:

```
from sklearn import cross_validation

Xucenje, Xtest, Yucenje, Ytest = cross_validation.train_test_split
    (data_set, klase, test_size=0.4, random_state=0)

clf.fit(x_ucenje, y_ucenje)

rezultat = cross_validation.cross_val_score(clf, x_test, y_test)
print "Točnost algoritma: %0.2f +/- %0.2f
    )" % (rezultat.mean(), rezultat.std() / 2)
```

5.5 Implementacija k-NN algoritma

Pythonov modul sklearn integrira klasične algoritme za strojno učenje s ciljem pružanja jednostavnog i efikasnog rješenja za probleme učenja. Na raspolaganju su algoritmi iz kategorije nadziranog i nenadziranog učenja te razni setovi podataka na kojima možemo isprobavati algoritme. Kako je spomenuto, k-NN algoritam najbolje radi pri malim dimenzijama, iz tog razloga izabrani su dvodimenzijalni vektori, osim toga lakše ih je predočiti na grafu te vizualizirati formiranje klastera. No, samo dvije značajke nisu dovoljne da bi se napravila značajna razlika između objekata pa se koristi ukupno sedam takvih vektora. Nakon što je upit klasificiran, tj. vraćeni rezultati algoritma, opet se kreče u prebrojavanje glasova, te je opet većinom glasova odabrana klasa ispitnog uzorka.

Na sljedećem primjeru biti će pokazano kako je zamišljena implementacija. Algoritam je naučen da prepoznaje dvije vrste objekta – trokut i pravokutnik. Prilikom učenja napravljene su razne varijacije tih objekta – skalirane, okrenute i translatirane za različite vrijednosti. Na slici 5.2 prikazane su jedne od varijanti objekata s kojima je algoritam učen.

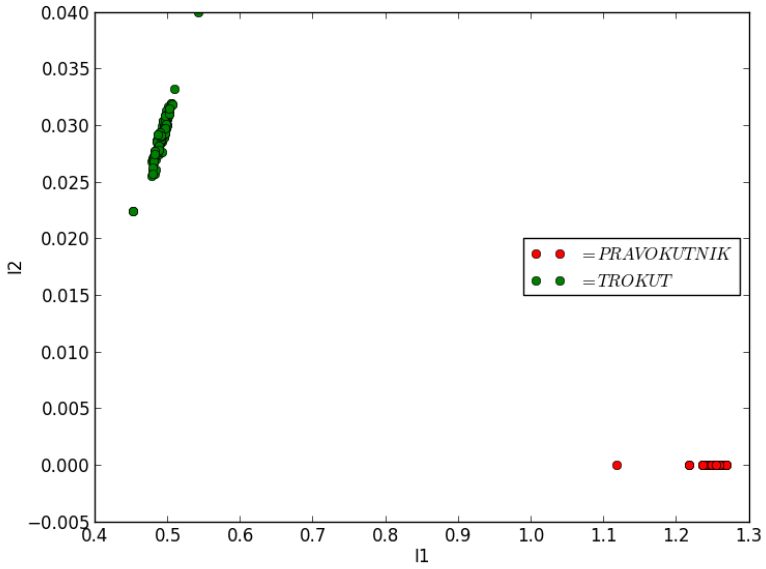


Slika 5.2: Objekti s kojima je učen algoritam.

Na slici 5.3 prikazan je rezultat učenja za afine momente I1 i I2. Slično izgledaju grafovi i za ostalih 12 momenata.

Podaci za učenje pripremljeni su tako što je u sliku veličine 350x350 piksela upisano napravio razne varijacije te slike tako što ju je rotirao i skalirao za nasumični broj stupnjeva i u nasumičnom omjeru. Za svaku sliku su izračunati momenti, te spremljeni u bazu. Baza je formirana tako da se svaka vrsta momenta spremi u svoju tablicu (ukupno pet tablica). Kako bi algoritam mogao automatski učiti iz baze potrebni su mu podaci o identifikaciji objekta i njegove značajke.

Ako napravimo upit za dani primjer, dobit ćemo sljedeći ispis:



Slika 5.3: Prikaz afinih momenata I1 i I2 za primjer trokuta i pravokutnika.

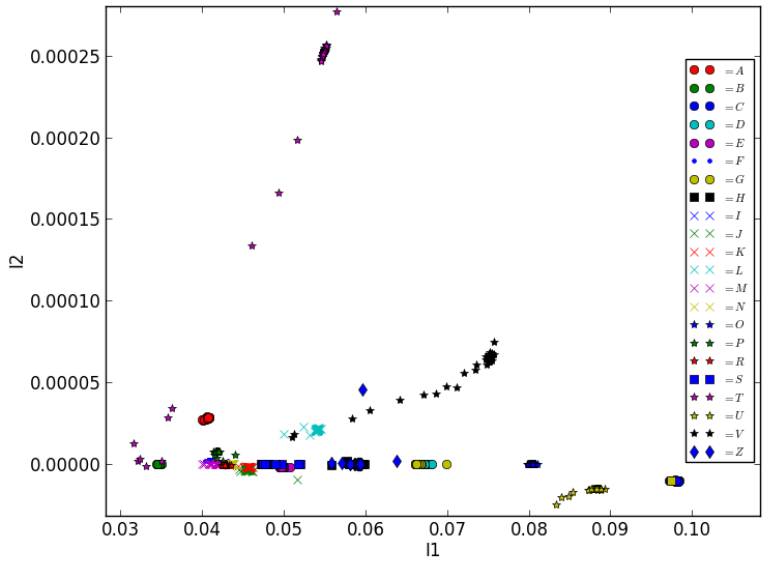
```
>>> Z1 = clf1.predict([moment1,  moment2])
>>> Z2 = clf2.predict([moment3,  moment4])
>>> Z3 = clf3.predict([moment5,  moment6])
>>> Z4 = clf4.predict([moment7,  moment8])
>>> Z5 = clf5.predict([moment9, moment10])
>>> Z6 = clf6.predict([moment11,moment12])
>>> Z7 = clf7.predict([moment13,moment14])
>>> print [Z1, Z2, Z3, Z4, Z5, Z6, Z7]
['p', 'p', 'p', 'p', 'p', 'p', 'p']
```

Navedeni primjer prikazuje da objekt pripada klasi `pravokutnik`.

Sljedeći primjer pokazuje bitno drugačiju situaciju. Na slici 5.4 vidi se kako su formirani klasteri za 22 velikih slova pisanih rukom. Određeni klasteri su jako blizu ili se preklapaju, a i raspršenost podataka je prilično velika. Idealno bi bilo kada bi bili klasteri bili separabilni kao u slučaju s trokutom i pravokutnikom. No, kako to nije moguće postići koristi se još šest parova značajki kako bi se odredilo kojem klasteru pripada objekt. Jasno je postoji mogućnost krive klasifikacije, zato su tu ostalih šest da bi se povećala vjerojatnost ispravne klasifikacije što ne isključuje mogućnost još uvijek krive klasifikacije. Ako bazu učenja napunimo s npr. više vrsti fontova doći će do stupnja preklapanje klastera gdje više nije moguća točna klasifikacije već pogađanje o kojem se objektu radi što čini određeno ograničenje ovog pristupa prepoznavanja.

```
>>> Z1 = clf1.predict([moment1,  moment2])
>>> Z2 = clf2.predict([moment3,  moment4])
>>> Z3 = clf3.predict([moment5,  moment6])
>>> Z4 = clf4.predict([moment7,  moment8])
>>> Z5 = clf5.predict([moment9, moment10])
>>> Z6 = clf6.predict([moment11,moment12])
>>> Z7 = clf7.predict([moment13,moment14])
>>> print [Z1, Z2, Z3, Z4, Z5, Z6, Z7]
['A', 'A', 'G', 'A', 'G', 'F', 'T']
```

Većinom glasova objekt je klasificiran kao slovo A.



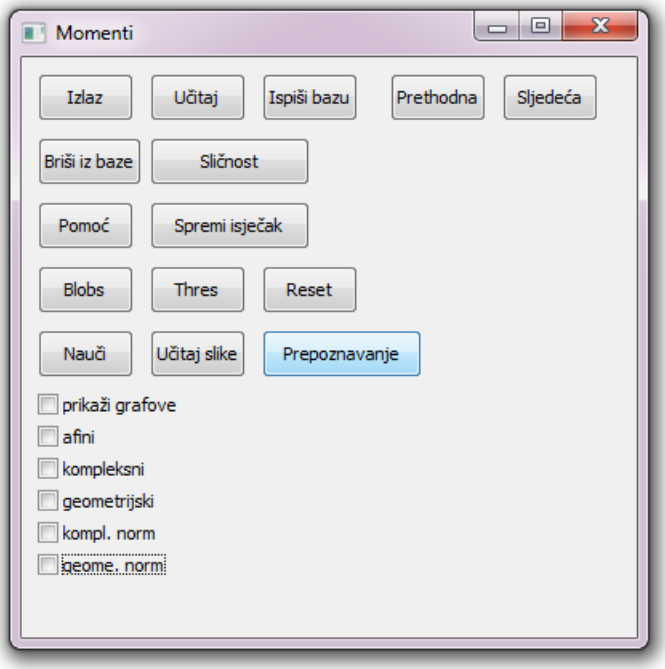
Slika 5.4: Prikaz afinih momenata I1 i I2 za primjer 22 velikih tiskanih slova.

OPIS PROGRAMSKOG RJEŠENJA

U ovom poglavlju biti će objašnjeno idejno rješenje programa te njegove funkcije koje su dostupne korisniku kroz sučelje.

6.1 Grafičko sučelje

Slika 6.1 prikazuje grafičko sučelje programa. Kroz njega su dostupne sve funkcionalnosti koje program pruža.



Slika 6.1: Izgled sučelja programa.

Funkcijske tipke biti će objašnjene redoslijedom kojim se pojavljuju u programu.

- *Izlaz* služi za izlaz iz programa.
- *Učitaj* učitava sliku (ili više njih) na kojima će biti vršena daljnja obrada.

- *Ispiši bazu* otvara novi prozor na kojem prikazuje što se nalazi u bazi. U bazu se spremaju slike odnosno njeni isječci na kojima je prethodno vršena neka obrada ili su računati momenti.
- *Prethodna* i *Sljedeća* služe samo ako je otvoreno više slika pa pomoću njih vršimo njihovo listanje.
- *Briši iz baze* možemo izbrisati bilo koju stavku iz baze.
- *Sličnost* otvara novi prozor u kojemu se traži da korisnik unese identifikacijski broj slike (identifikacijski broj svake slike je vidljiv kad korisnik pregledava bazu - *Ispiši bazu*). Ono što se događa u pozadini je da se vrši prepoznavanje objekta na slici pomoću odabranih invarijantnih momenata, te se ispisuje rezultat upita.
- *Pomoć* otvara novi prozor u kojem su opisane funkcionalnosti programa.
- *Spremi isječak* sprema odabrani objekt sa slike nakon što smo ju obradili na željeni način.
- *Thres* vrši threshold slike.
- *Blobs* - nakon što je threshold slike izvršen te odabrana željena vrijednost thresholda, moguće je izvršiti funkciju prepoznavanja blobova na slici. Blobovi su skupina piksela koji su međusobno povezani te čine cjelinu. Određene blobove želimo filtrirati sa slike bilo da su dio šuma koji dolazi zbog načina na koji je slika napravljena ili jednostavno blobovi ne čine dio objekta . Pomoću klizača koji se pojave nakon što su blobovi prepoznati moguće je izvršiti spomenutu filtraciju.
- *Reset* vraća sliku u početno stanje bez ikakvih prethodnih promjena koje su eventualno bile izvršene.
- *Nauči* nam služi da bi upotpunili bazu učenja novom stavkom ako je taj objekt bio krivo prepoznat u prethodnom upitu te na taj način vršimo korekciju.
- *Učitaj slike* se može upotrijebiti za brzo punjenje baze učenja. Potrebno je prije toga imati unaprijed pripremljene slike s kojima želimo učiti algoritam. Na svakoj slici mora biti samo jedan objekt te će program automatski izvršiti threshold na njima, a ime objekta moguće je unijeti ručno za svaku sliku ili jednostavno prepustiti da program automatski prepozna iz imena slike o kojem se objektu radi.
- *Prepoznavanje* služi kao funkcija za automatsko prepoznavanje objekata na slici. Prije nego možemo upotrijebiti funkciju automatskog prepoznavanja potrebno je na slici napraviti threshold, te detektirati blobove. Rezultati prepoznavanja bit će spremljeni u bazu, a pregledavanjem baze mogu se analizirati rezultati.

Pomoću liste na dnu prozora odabiremo koje momente želimo koristiti pri prepoznavanju te da li želimo vidjeti grafove za momente. Na grafovima je vidljivo gdje se nalazi naš upit iz bazu

u odnosu na druge klastere te zašto je objekt eventualno dobro ili loše prepoznat.

6.2 Ključni dijelovi programa

Program je realiziran kroz četiri razreda. Razredi su podijeljeni prema cjelinama na koje se odnose:

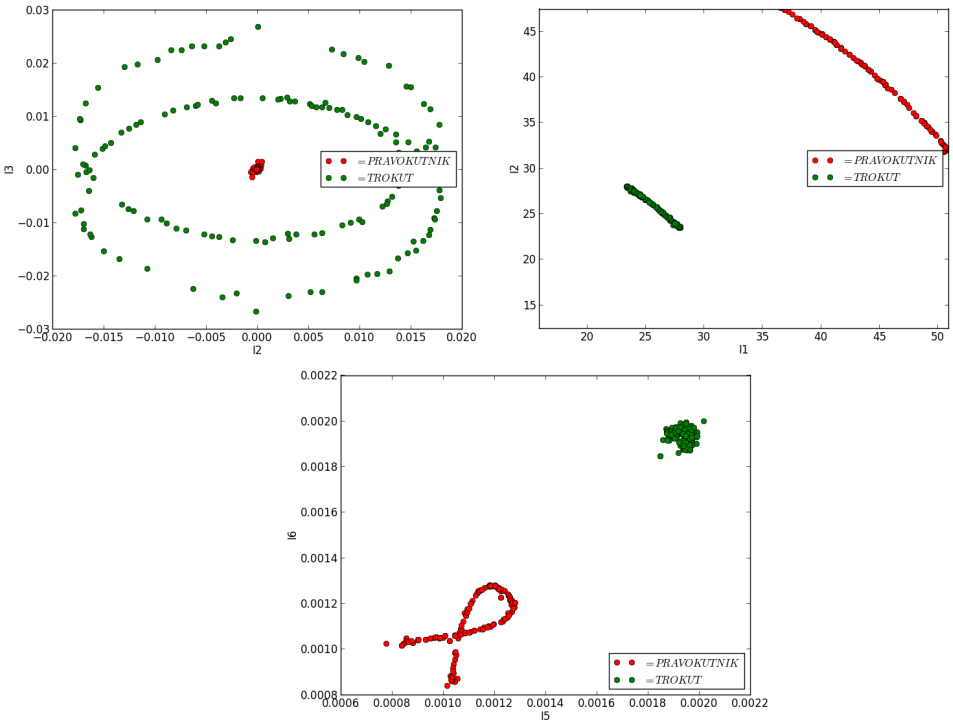
- *Grafsucelje* je razred koji brine o grafičkom sučelju programa i ostalim interakcijama s korisnikom.
- *Slika* je razred kroz koji se vrše baratanja sa slikom. Tu je uključeno od prikaza slike na ekranu do krajnje obrade.
- *Moment* je razred koji prilikom stvaranja svojeg objekta mora imati proslijeđen objekt za koji izračunava invarijantne momente.
- Razred *Baza* se koristi prilikom svih baratanja s bazom, a to su upiti u bazu, spremanje stavki, pretraga baze te ostale funkcionalnosti bitne za rad baze.

Kao što je bilo navedeno u poglavlju o razredima, korištenjem razreda struktura kôda je puno logičnija i lakše se snalaziti. Također, kad je kôd organiziran u razrede moguće je i njegovo ponovno korištenje u sličnim aplikacijama.

EKSPERIMENTALNI PODACI

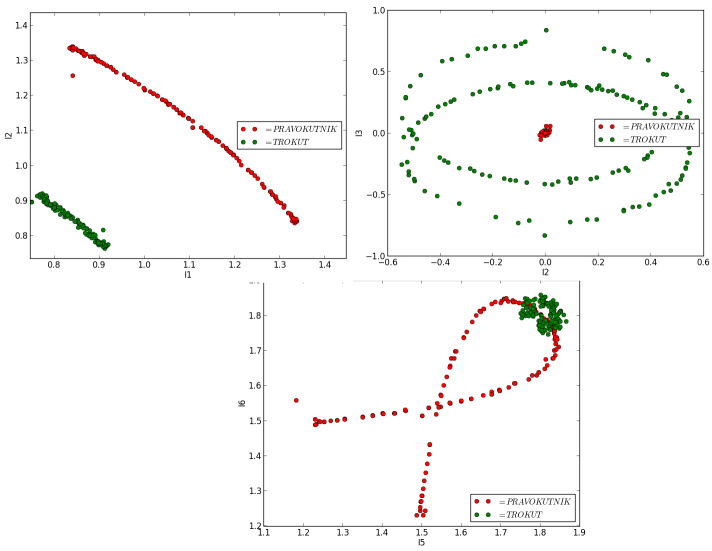
7.1 Ispitivanja na objekte - trokut, pravokutnik

Invarijantni geometrijski, normalizirani geometrijski, kompleksni i normalizirani kompleksni momenti ispitani su jednostavnim objektima – trokut i pravokutnik. Navedeni momenti su izračunati za objekte koji se razlikuju u poziciji i rotaciji na slici.



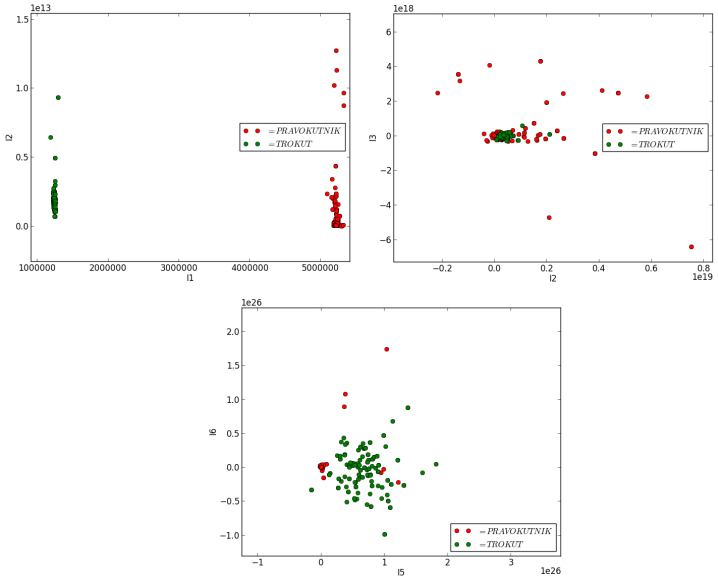
Slika 7.1: Geometrijski momenti za objekte trokut i pravokutnik.

Geometrijski momenti imaju dobru karakteristiku za navedeni slučaj već samim time što su klasteri odvojeni te time je napravljen preduvjet da je moguće razlikovati objekte. Dakako, mana im je što nisu invarijantni na skaliranje, no ako to nije ni uvijet koji mora biti zadovoljen ovi momenti mogu biti dobar izbor. Iako nisu invarijantni na skaliranje u testovima za ova dva objekta prepoznavanje je bilo točno za sve zadane primjere.



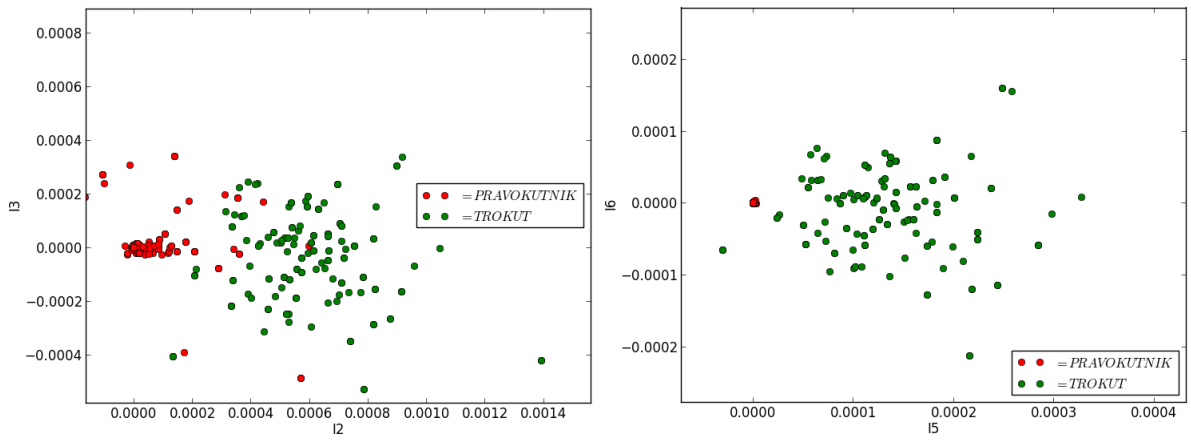
Slika 7.2: Normalizirani geometrijski momenti za objekte trokut i pravokutnik.

Normalizirani geometrijski momenti ispravljaju manu geometrijskih momenata. Normalizacijom se izmjenila karakteristika momenata. Testovi napravljeni s normaliziranim geometrijskim momentima opet su dali odlične rezultate.



Slika 7.3: Kompleksni momenti za objekte trokut i pravokutnik.

Kompleksni momenti imaju relativno dobru karakteristiku, odnosno klasteri su na neki način odvojeni ali ne onako kako smo vidjeli kod geometrijskih momenata. Relativno dobru u smislu da je dobra ako će momenti izračunati za zadani objekt uvijek padati u te točke. No, pokazalo se da to nije tako. Kompleksni momenti nisu bili u mogućnosti prepoznati točno objekte. Svi objekti redom su spadali u kategoriju pravokutnika te tako nisu prepoznali niti jedan zadani trokut.

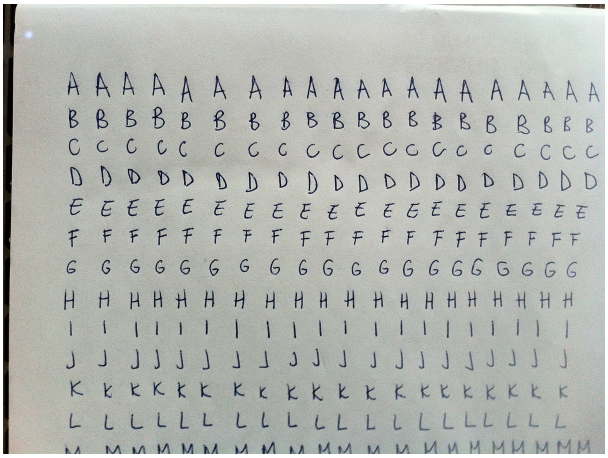


Slika 7.4: Normalizirani kompleksni momenti za objekte trokut i pravokutnik.

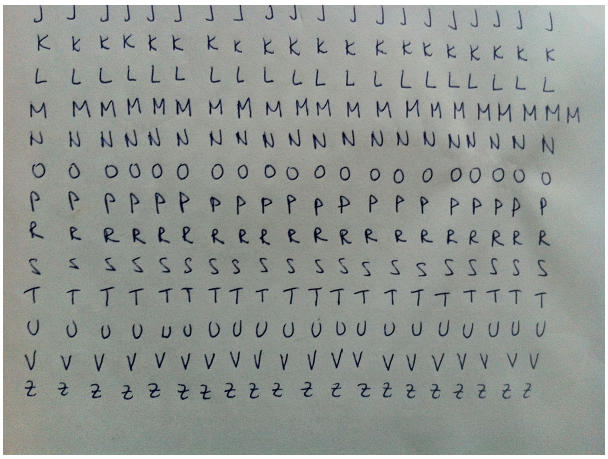
Kod normaliziranih kompleksnih momenata situacija je nešto drugačija nego kod kompleksnih. Bili su u stanju prepoznati većinu zadanih trokut objekta i sve pravokutnike. Objekti crtani rukom su također točno prepoznati. Problem je što smo u ovom slučaju izgubili invarijante zbog normalizacije te je time smanjena diskriminacijska moć.

7.2 Ispitivanja na rukopisu

Napravljeno je testiranje prepoznavanje slova (samo velikih tiskanih) napisanih rukom. Tu dolazi do niza problema jer svako slovo nije isto, pa je potrebna velika baza učenja raznih varijacija slova. No, nije samo u tome problem, drugačija segmentacije (npr. odabrana vrijednost thresholda) također može utjecati na točnost prepoznavanja. Kada govorimo o segmentaciji dotičemo se i načina uzimanja slike (slikanjem dokumenta fotoaparatom ili skeniranjem). Također tamo gdje je trag pri pisanju slova na papir slabije ostavljen utječe na točnost pri prepoznavanju. Na slici 7.5 i slici 7.6 je prikazan papir na kojem su slova ispisana rukom i s kojima se krenulo u fazu učenja.



Slika 7.5: Slova s kojima se krenulo u fazu učenja.



Slika 7.6: (nastavak) Slova s kojima se krenulo u fazu učenja.

Nakon što je svako slovo uvršteno u bazu napravljene su i njegove varijacije sa zakretanjem za različite kutove i skaliranja u različitim mjerama u svrhu dobivanja raznovrsnijih podataka. Na slici 7.7 su prikazana slova nakon segmentacije.

Slika 7.7: Slova nakon segmentacije.

U tablici su prikazani rezultati prepoznavanja za dani slučaj.

Tablica 7.1: Rezultati prepoznavanja rukom pisanih slova pomoću afinih momenata.

Slovo	Prepoznato	Postotak prepoznatih
A	8/20	40%
B	6/20	30%
C	18/20	90%
D	1/20	5%
E	7/18	38%
F	5/20	25%
G	2/15	13%
H	5/19	26%
I	10/10	100%
J	4/20	20%

Tablica 7.2: (nastavak) Rezultati prepoznavanja rukom pisanih slova pomoću afinih momenata.

Slovo	Prepoznato	Postotak prepoznatih
K	3/20	15%
L	3/20	15%
M	13/20	65%
N	3/20	15%
O	3/20	15%
P	2/20	10%
R	7/20	35%
S	2/20	10%
T	16/20	80%
U	2/20	10%
V	2/20	10%
Z	8/20	40%
Ukupno	130/442	29%

Kako su momenti invarijantni na rotaciju, moguće su određene zamjene slova za neka druga kada su rotirana. Odnosno, čak i za ljude je moguća kriva klasifikacija kada su slova pisana rukom. Tako slovo 'U' u određenim situacijama može izgledati kao rotirano slovo 'C' i obrnuto. Sve ovisi u kojem se kontekstu promatra određena scena. Slika 7.8 prikazuje takav slučaj.

Slika 7.8: Primjer ovisnosti klasifikacije o kontekstu.

Ovisno o načinu na koji čitamo sliku 7.8 pročitati ćemo ili "ABC" (s tim da je slovo B napisano malo čudno) ili "132".

7.3 Ispitivanja tiskanih slova

Ispitivanja tiskanih slova napravljeno je tako da je na sliku napisano 22 velikih tiskanih slova veličine 180pt. Nakon što je izvršena segmentacija, krenulo se u fazu prepoznavanja. Očekivano, rezultati su bili da su sva slova točno prepoznata. Unatoč tome što dolazi do preklapanja klastera točnost je velika, a razlog leži u tome što su slova tiskana na računalu uvijek ista. Stoga nema greški u segmentaciji, te su momenti praktički jednaki. Napravljen je isti test s veličinama slova 80pt i 40 pt gdje su rezultati bili značajno drugačiji. Točnost prepoznavanja je drastično pala. Stopa prepoznavanja je pala na 13,63% i 27,27% respektivno. Problem je što invarijantnost na skaliranje nije jedinstvena već slova postaju dosta slično kada ih se smanji za 3-4 puta.

ZAKLJUČAK

U ovom diplomskom radu dan je pregled invarijantnih momenata kao opisivača oblika. Invarijantni momenti postoje već dugo, a ovdje su dana uvodna razmatranja. Dok invarijante na translaciju, rotaciju i skaliranje su poznati dug niz godina, invarijante na afine transformacije su relativno nedavno razvijene. Dakle, momenti su globalne značajke, izračunate iz cijele slike ili određene regije slike. Njihova globalnost rezultira lošom robusnošću na neočekivanu poremećaje na slici ili individualne oblike objekta, respektivno. Kada objekt nije u cijelosti prikazan u kadru je tipičan primjer. Čak i male lokalne promjene na slici mogu utjecati na sve momente slike. Stoga, odabir regije od interesa i njena valjana lokalizacija je bitna za uspješnu primjenu momenata. Momenti su osjetljivi na greške segmentacije (loše prepoznavanje slova pisanih rukom), zato segmentacijski algoritam mora biti pažljivo odabran. Rezultati provedeni za ovaj rad nisu toliko dobri kako se očekivalo ali mogu biti polazna točka za daljnja razmatranja, te ulazak u područje računalne vizije. Naravno, postoje i određena područja njihove primjene gdje rade odlično. Za ovaj rad napravljen je program koji omogućuje testiranje invarijantnih momenata, njihovu evaluaciju i analizu rezultata.

Literatura

[1] Gary Bradski and Adrian Kaehler. *Learning OpenCV*. Packt Publishing Ltd., 2008. ISBN 978-0-596-51613-0.

[2] Hiromichi Fujisawa Cheng-Lin Liu. Classification and learning for character recognition: Comparison of methods and remaining problems.

[3] Ioan Snep Dan L. Lacrama. Moment invariants in image analysis.

[4] Ioan Snep Dan L. Lacrama. The use of invariant moments in hand-written character recognition.

[5] Jan Flusser. Moment invariants in image analysis.

[6] Jan Flusser. On the independence of rotation moment invariants, 1999.

[7] Barbara Zitova Jan Flusser, Tomas Suk. *Moments and Moment Invariants in Pattern Recognition*. John Wiley Sons Ltd, 2009. ISBN 978-0-470-69987-4.

[8] Tomas Suk Jan Flusser. Tables of affine moment invariants generated by the graph method.

[9] G.N. Skhar K.R. Radhika, M.K. Venkatesha. Off-line signature authentication based on moment invariants using support vector machine, 2010.

[10] Robert Laganiere. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing Ltd., 2011. ISBN 978-1-849513-24-1.

[11] Simon Xinmeng Liao. Image analysis by moments. doctoral thesis, 1993.

[12] Jan Flusser Tomas Suk. On the independence of rotation moment invariants.

Dodatak

```
from __future__ import division
import os
import sqlite3
import cv2 as cv
import wx
import numpy
import math
import cmath
import random
import matplotlib.pyplot as plt
from sklearn import neighbors
############################

class Moment():
    def __init__(self, image):
        self.img = image.copy()

        self.n1, self.n2 = image.shape
        self.img = image
        self.img = cv.bitwise_not(self.img)
        self.afinvektor = self.afini()
        self.geometrijski = self.VektorCentralGeometricMoments()
        self.kompleksni = self.VektorComplexMoments()
        self.skompleksni = self.VektorScaleComplexMoments()
        self.sgeometrijski = self.VektorScaleCentralGeometricMoments()

    def pow_complex(self, real, imag, n):
        (r, phi) = cmath.polar(complex(real,imag))
        r_n = math.pow(r,n)
        z = complex(r_n * math.cos(n*phi), r_n * math.sin(n*phi))
        return z

    def Moment_pq(self, p, q):
        sum1 = 0
        for i in range(self.n1):
            for j in range(self.n2):
```

```
                if(self.img.item(i,j) >= 120):
                    sum1 += math.pow(i,p) * math.pow(j,q)
        return sum1

    def CentralComplex_pq(self, p, q):
        m00 = self.Moment_pq(0, 0)
        m10 = self.Moment_pq(1, 0)
        m01 = self.Moment_pq(0, 1)

        mx1 = m10/m00
        my1 = m01/m00

        sum1=0
        for i in range(self.n1):
            for j in range(self.n2):
                if(self.img.item(i,j) == 1):
                    sum1 += self.pow_complex( (i-mx1), (j-my1), p) * self.
                        pow_complex( (i - mx1), -(j - my1), q)
        return sum1

    def ScaleCentralComplex_pq(self, p, q):
        m00 = self.Moment_pq(0, 0)
        m10 = self.Moment_pq(1, 0)
        m01 = self.Moment_pq(0, 1)

        mx1 = m10/m00
        my1 = m01/m00

        sum1=0
        for i in range(self.n1):
            for j in range(self.n2):
                if(self.img.item(i,j) == 1):
                    sum1 += self.pow_complex( (i-mx1), (j-my1), p) * self.
                        pow_complex( (i - mx1), -(j - my1), q)

        return sum1/(math.pow(m00,((p+q)/2 + 1)))

    def VektorScaleComplexMoments(self):
        c12 = self.ScaleCentralComplex_pq(1,2)
        c20 = self.ScaleCentralComplex_pq(2,0)
        c30 = self.ScaleCentralComplex_pq(3,0)
        F20 = c20*self.pow_complex(c12.real, c12.imag, 2)
        F30 = c30*self.pow_complex(c12.real, c12.imag, 3)

        return [0, 0, F20.real, F20.imag, F30.real, F30.imag]
```

```
    def VektorComplexMoments(self):
        c11 = self.CentralComplex_pq(1,1)
        c21 = self.CentralComplex_pq(2,1)
        c12 = self.CentralComplex_pq(1,2)
        c20 = self.CentralComplex_pq(2,0)
        c30 = self.CentralComplex_pq(3,0)

        F11 = c11
        F21 = c21*c12
        F20 = c20*self.pow_complex(c12.real, c12.imag, 2)
        F30 = c30*self.pow_complex(c12.real, c12.imag, 3)

        return [F11.real, F21.real, F20.real, F20.imag, F30.real, F30.imag]

    def CentralGeometricMoments(self, p, q):
        m00 = self.Moment_pq(0, 0)
        m10 = self.Moment_pq(1, 0)
        m01 = self.Moment_pq(0, 1)

        mx1 = m10/m00
        my1 = m01/m00

        sum1=0
        for i in range(self.n1):
            for j in range(self.n2):
                if(self.img.item(i,j) == 1):
                    sum1 +=  (i - mx1)**p * (j - my1)**q
        return sum1

    def ScaleCentralGeometricMoments(self, p, q):
        m00 = self.Moment_pq(0, 0)
        m10 = self.Moment_pq(1, 0)
        m01 = self.Moment_pq(0, 1)


        mx1 = m10/m00
        my1 = m01/m00

        sum1=0
        for i in range(self.n1):
            for j in range(self.n2):
                if(self.img.item(i,j) == 1):
                    sum1 +=  (i - mx1)**p * (j - my1)**q
        return sum1/(math.pow(m00,((p+q)/2 + 1)))

    def VektorCentralGeometricMoments(self):
        c00 = self.CentralGeometricMoments(0,0)
```

```
        c20 = self.CentralGeometricMoments(2,0)
        c02 = self.CentralGeometricMoments(0,2)
        c30 = self.CentralGeometricMoments(3,0)
        c03 = self.CentralGeometricMoments(0,3)
        c40 = self.CentralGeometricMoments(4,0)
        c04 = self.CentralGeometricMoments(0,4)

        w20 = math.sqrt(c20/c00)
        w02 = math.sqrt(c02/c00)

        w30 = c30/(math.sqrt(math.pow(c20, 3)))
        w03 = c03/(math.sqrt(math.pow(c20, 3)))

        w40 = c40/(c20**2)
        w04 = c04/(c02**2)

        return [w20, w02, w30, w03, w40, w04]

    def VektorScaleCentralGeometricMoments(self):
        c00 = self.ScaleCentralGeometricMoments(0,0)
        c20 = self.ScaleCentralGeometricMoments(2,0)
        c02 = self.ScaleCentralGeometricMoments(0,2)
        c30 = self.ScaleCentralGeometricMoments(3,0)
        c03 = self.ScaleCentralGeometricMoments(0,3)
        c40 = self.ScaleCentralGeometricMoments(4,0)
        c04 = self.ScaleCentralGeometricMoments(0,4)

        w20 = math.sqrt(c20/c00)
        w02 = math.sqrt(c02/c00)

        w30 = c30/(math.sqrt(math.pow(c20, 3)))
        w03 = c03/(math.sqrt(math.pow(c20, 3)))

        w40 = c40/(c20**2)
        w04 = c04/(c02**2)

        return [w20, w02, w30, w03, w40, w04]

    def afini(self):
        m00 = self.CentralGeometricMoments(0, 0)

        m11 = self.CentralGeometricMoments(1, 1)
        m20 = self.CentralGeometricMoments(2, 0)
        m02 = self.CentralGeometricMoments(0, 2)
        m21 = self.CentralGeometricMoments(2, 1)
        m22 = self.CentralGeometricMoments(2, 2)
        m12 = self.CentralGeometricMoments(1, 2)
```

```
m30 = self.CentralGeometricMoments(3, 0)
m31 = self.CentralGeometricMoments(3, 1)
m13 = self.CentralGeometricMoments(1, 3)
m03 = self.CentralGeometricMoments(0, 3)
m40 = self.CentralGeometricMoments(4, 0)
m04 = self.CentralGeometricMoments(0, 4)
m05 = self.CentralGeometricMoments(0, 5)
m50 = self.CentralGeometricMoments(5, 0)
m41 = self.CentralGeometricMoments(4, 1)
m14 = self.CentralGeometricMoments(1, 4)
m32 = self.CentralGeometricMoments(3, 2)
m23 = self.CentralGeometricMoments(2, 3)

try:
    I1 = (m20*m02 - m11**2)/m00**4
    I2 = (-m30**2 * m03**2 + 6 * m30 * m21 * m12 * m03 - 4* m30 *
       m12**3 - 4 * m21**3 * m03 + 3 * m12**2 * m21**2)/m00**10
    I3 = (m20 * m21 * m03 - m20 * m12**2 - m11 * m30 * m03 + m11 *
       m21 * m12 + m02 * m30 * m12 - m02 * m21**2)/m00**7
    I4 = (-m20**3 * m03**2 + 6 * m20**2 *m11 * m12 * m03 - 3 * m20
       **2 * m02 * m12**2 - 6* m20 *m11**2 * m21 * m03 - 6 * m20 *
       m11**2 * m12**2 + 12 * m20 * m11 *m02 * m21 * m12 - 3 * m20
       * m02**2 * m21**2 + 2 * m11**3 * m30 * m03 + 6 * m11**3 *
       m21 * m12 - 6 * m11**2 * m02 * m30 * m12 - 6 * m11**2 * m02
       * m21**2 + 6 *m11 * m02**2 * m30 * m21 - m02**3 * m30**2)/
       m00**11
    I5 = (m20**3 * m30 * m03**3 - 3 * m20**3 * m21 * m12 * m03**2 +
        2 * m20**3 * m12**3 * m03 - 6 * m20**2 * m11 * m30 * m12 *
       m03**2 + 6 * m20**2 * m11 * m21**2 * m03**2 + 6 *m20**2 *
       m11 * m21 * m12**2 * m03 - 6 * m20**2 * m11 * m12**4 + 3 *
       m20**2 * m02 * m30 * m12**2 * m03 - 6 * m20**2 * m02 * m21
       **2 * m12 * m03 + 3 * m20**2 * m02 * m21 * m12**3 + 12* m20
       * m11**2 * m30 * m12**2 * m03 - 24 * m20 * m11**2 *m21**2 *
       m12 * m03 + 12 * m20 * m11**2 * m21 * m12**3 - 12  * m20 *
       m11 * m02 * m30 * m12**3 + 12 * m20 * m11 * m02 * m21**3 *
       m03 - 3 * m20 * m02**2 * m30 * m21**2 * m03 + 6 * m20 * m02
       **2 * m30 * m21 * m12**2 - 3 * m20 * m02**2 * m21**3 * m12 -
        8 * m11**3 * m30 * m12**3 + 8 * m11**3 * m21**3 * m03-12 *
       m11**2 * m02 * m30 * m21**2 * m03 + 24 * m11**2 * m02 * m30
       * m21 * m12**2 - 12 * m11**2 * m02 * m21**3 * m12+  6 * m11
       *m02**2 * m30**2 * m21 * m03 - 6*m11 * m02**2 * m30**2 * m12
       **2 - 6 * m11 * m02**2 * m30 * m21**2 * m12+ 6 * m11 * m02
       **2 * m21**4 - m02**3 * m30**3 * m03 + 3 * m02**3 * m30**2 *
        m21 * m12 - 2 * m02**3 * m30 * m21**3)/m00**16
    I6 = (m40 * m04 - 4 * m31 * m13 + 3 * m22**2)/m00**6
    I7 = (m40 * m22 * m04 - m40 * m13**2 - m31**2 * m04 + 2 * m31 *
        m22 * m13 - m22**3)/m00**9
```

```
I8 = (m20**2 * m04 - 4* m20* m11 * m13 + 2 * m20 * m02 * m22 +
   4 * m11**2 * m22 - 4 * m11 * m02 * m31 + m02**2 * m40)/m00
   **7
I9 = (m20**2 * m22 * m04 - m20**2 * m13**2 - 2 * m20 * m11* m31
    * m04 + 2 * m20 * m11 * m22 * m13 + m20 * m02 * m40* m04 -
   2 * m20 * m02 * m31 * m13 + m20 * m02 * m22**2 + 4 * m11**2
   * m31 * m13 - 4* m11**2 * m22**2 - 2 * m11 * m02 * m40 * m13
    + 2 * m11 * m02 * m31 * m22 + m02**2 * m40 * m22 - m02**2 *
    m31**2)/m00**10

I10= (m20**3 * m31 * m04**2 - 3*m20**3*m22*m13*m04 + 2*m20**3*
   m13**3 - m20**2*m11*m40*m04**2 - 2 * m20**2*m11*m31*m13*m04
   + 9*m20**2*m11*m22**2*m04 - 6*m20**2*m11*m22*m13**2+m20**2*
   m02*m40*m13*m04 - 3*m20**2*m02*m31*m22*m04 + 2*m20**2*m02*
   m31*m13**2+4*m20*m11**2*m40*m13*m04 - 12*m20*m11**2*m31*m22*
   m04 + 8*m20*m11**2*m31*m13**2-6*m20*m11*m02*m40*m13**2 + 6*
   m20*m11*m02*m31**2*m04 - m20*m02**2*m40*m31*m04+3*m20*m02
   **2*m40*m22*m13 - 2*m20*m02**2*m31**2*m13 - 4*m11**3*m40*m13
   **2 + 4*m11**3*m31**2*m04-4*m11**2*m02*m40*m31*m04 + 12*m11
   **2*m02*m40*m22*m13 - 8*m11**2*m02*m31**2*m13 + m11*m02**2*
   m40**2*m04 + 2*m11*m02**2*m40*m31*m13 - 9*m11*m02**2*m40*m22
   **2 + 6*m11*m02**2*m31**2*m22 - m02**3*m40**2*m13 + 3*m02
   **3*m40*m31*m22 - 2*m02**3*m31**3)/m00**15

I11= (m30**2*m12**2*m04 - 2*m30**2*m12*m03*m13 + m30**2*m03**2*
   m22 - 2*m30*m21**2*m12*m04 + 2*m30*m21**2*m03*m13 + 2*m30*
   m21*m12**2*m13 - 2*m30*m21*m03**2*m31 - 2*m30*m12**3*m22 +
   2*m30*m12**2*m03*m31 + m21**4*m04 - 2*m21**3*m12*m13 - 2*m21
   **3*m03*m22 + 3*m21**2*m12**2*m22 + 2*m21**2*m12*m03*m31 +
   m21**2*m03**2*m40 - 2*m21*m12**3*m31 - 2*m21*m12**2*m03*m40
   + m12**4*m40)/m00**13
I12= (m30**2*m31*m04**2 - 3*m30**2*m22*m13*m04 + 2*m30**2*m13
   **3 - m30*m21*m40*m04**2-2*m30*m21*m31*m13*m04 + 9*m30*m21*
   m22**2*m04 - 6*m30*m21*m22*m13**2+2*m30*m12*m40*m13*m04 - 6*
   m30*m12*m31*m22*m04 + 4*m30*m12*m31*m13**2-m30*m03*m40*m13
   **2 + m30*m03*m31**2*m04 + 3*m21**2*m40*m13*m04 - 9*m21**2*
   m31*m22*m04+6*m21**2*m31*m13**2 - 9*m21*m12*m40*m13**2 + 9*
   m21*m12*m31**2*m04 - 2*m21*m03*m40*m31*m04+6*m21*m03*m40*m22
   *m13 - 4*m21*m03*m31**2*m13 - 3*m12**2*m40*m31*m04+9*m12**2*
   m40*m22*m13 - 6*m12**2*m31**2*m13 + m12*m03*m40**2*m04 + 2*
   m12*m03*m40*m31*m13-9*m12*m03*m40*m22**2 + 6*m12*m03*m31**2*
   m22 - m03**2*m40**2*m13 + 3*m03**2*m40*m31*m22-2*m03**2*m31
   **3)/m00**14
I19= (m20 * m30 * m12 * m04 - m20 * m30 * m03 * m13 - m20 * m21
   **2 * m04 + m20 * m21 * m12 * m13 + m20 * m21 * m03 * m22 -
   m20 * m12**2 * m22 - 2 * m11 * m30* m12 * m13 + 2 * m11 *
   m30 * m03 * m22 + 2 * m11 * m21**2 * m13 - 2 * m11 * m21 *
```

```
            m12 * m22 - 2 * m11 * m21 * m03 * m31 + 2 * m11 * m12**2 *
            m31 + m02 * m30 * m12 * m22 - m02 * m30 * m03 * m31 - m02 *
            m21**2 * m22 + m02 * m21 * m12 * m31 + m02 * m21 * m03 * m40
             - m02 * m12**2 * m40)/m00**10
        I47= (-m50**2 * m05**2 + 10 * m50 * m41 * m14 * m05 - 4 * m50 *
             m32 * m23 * m05 - 16 * m50 * m32 * m14**2 + 12 * m50 * m23
            **2 * m14 - 16 * m41**2 * m23 * m05 - 9 * m41**2 * m14**2 +
            12* m41 * m32**2 * m05 + 76 * m41 * m32 * m23 * m14 - 48 *
            m41 * m23**3 - 48 * m32**3 * m14 + 32 * m32**2 * m23**2)/m00
            **14

        return [I1, I2, I3, I4, I5, I6, I7, I8, I9, I10, I11, I12, I19,
             I47]
    except ZeroDivisionError:
        print "dijeljenje s nulom"
        return [0,0,0,0,0,0,0,0,0,0,0,0,0]
```